LPI exam 101 prep: Linux installation and package management

Junior Level Administration (LPIC-1) topic 102

Skill Level: Introductory

Ian Shields (ishields@us.ibm.com)
Senior Programmer
IBM

09 Sep 2005

In this tutorial, Ian Shields continues preparing you to take the Linux Professional Institute® Junior Level Administration (LPIC-1) Exam 101. In this second of five tutorials, Ian introduces you to Linux™ installation and package management. By the end of this tutorial you will know how Linux uses disk partitions, how Linux boots, and how to install and manage software packages.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at junior and intermediate levels. To attain each level of certification, you must pass two LPI exams. Before you take the exams, study the LPI exam prep tutorials on developerWorks to prepare for each topic covered in the exams.

The following five tutorials help you prepare for the first of the two LPI junior-level system administrator exams: LPI exam 101. A companion series of tutorials is under development for the other junior-level exam: LPI exam 102. Both exam 101 and exam 102 are required for junior-level certification. Junior-level certification is also known as certification level 1. To pass level 1, you should be able to:

- Work at the Linux command line

- Perform easy maintenance tasks: help out users, add users to a larger system, back up and restore, and shut down and reboot
- Install and configure a workstation (including X) and connect it to a LAN, or connect a stand-alone PC via modem to the Internet

Table 1. LPI exam 101: Tutorials and topics

Tutorial	Topic	Summary
LPI exam 101 prep: Hardware and architecture	Topic 101	Learn to configure your system hardware with Linux. By the end of this tutorial, you will know how Linux configures the hardware found on a modern PC and where to look if you have problems.
LPI exam 101 prep: Linux installation and package management	Topic 102	(This tutorial) Get an introduction to Linux installation and package management. By the end of this tutorial, you will know how Linux uses disk partitions, how Linux boots, and how to install and manage software packages.
LPI exam 101 prep: GNU and UNIX commands	Topic 103	Coming soon!
LPI exam 104 prep: Linux, filesystems, and FHS	Topic 104	Coming soon!
LPI exam 110 prep: The X Window system	Topic 110	Coming soon!

Within each exam topic, subtopics are weighted, reflecting their importance within the topic.

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Linux installation and package management", the second of five tutorials designed to prepare you for LPI exam 101. In this tutorial you will learn about these areas of Linux installation and package management:

Table 2. LPI exam 101, topic 102: Subtopics and weights

Subtopic	Weight	Summary
1.102.1 Design hard disk layout	5	You will learn to design a disk partitioning scheme for a Linux

		system, including allocating filesystems or swap space to separate partitions or disks, and tailoring the design to the intended use of the system. You will learn how to place /boot on a partition that conforms to BIOS requirements for booting.
1.102.2 Install a boot manager	1	You will learn to select, install, and configure a boot manager. You will learn how to provide alternative boot locations and backup boot options (for example, using a boot floppy).
1.102.3 Make and install programs from source	5	You will learn to build and install an executable program from source. You will learn to unpack a file of sources and customize the Makefile, for example to change paths or add extra include directories.
1.102.4 Manage shared libraries	3	You will learn to determine the shared libraries that executable programs depend on and install them when necessary. You will also learn where system libraries are kept.
1.102.5 Use Debian package management	8	You will learn to perform package management using the Debian package manager. You will use command-line and interactive tools to install, upgrade, or uninstall packages, as well as find packages containing specific files or software. You will learn how to determine package information such as version, content, dependencies, package integrity, and installation status. You will learn how to do this for both installed packages and packages that are not installed.
1.102.6 Use Red Hat Package Manager (RPM)	8	You will learn to perform package management for Linux distributions that use RPMs for package distribution. You will be able to install, re-install, upgrade, and remove packages, as well as obtain status and version information on packages. You

	will learn how to determine package information such as version, status, dependencies, integrity, and signatures. You will know how to to determine what files a package provides, as well as find which package supplies a specific file.

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial. You should also understand the BIOS implications for hard drives as covered in the first tutorial in this series, "LPI exam 101 prep (topic 101): Hardware and architecture."

Section 2. Hard disk layout

This section covers material for topic 1.102.1 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 5.

This section shows you how to lay out your Linux filesystems on your hard drives, and expands on what you learned about hard disks in the first tutorial in this series, "LPI exam 101 prep (topic 101): Hardware and architecture."

An upcoming tutorial, "LPI exam 101 prep (topic 104): Devices, Linux filesystems, and FHS" goes into more depth on filesystems and tools to create partitions of various types.

Filesystem overview

A Linux filesystem contains *files* that are arranged on a disk or other *block storage device* in *directories*. As with many other systems, directories on a Linux system may contain other directories called *subdirectories*. Unlike a system such as Microsoft® Windows® with a concept of separate file systems on different drive letters (A:, C:, etc.), a Linux filesystem is a single tree with the / directory as its *root* directory.

You might wonder why disk layout is important if the filesystem is just one big tree. Well, what really happens is that each block device, such as a hard drive partition, CD-ROM, or floppy disk, actually has a filesystem on it. You create the single tree view of the filesystem by *mounting* the filesystems on different devices at a point in the tree called a *mount point*.

Usually, you start this mount process by mounting the filesystem on some hard drive partition as /. You may mount other hard drive partitions as /boot, /tmp, or /home. You may mount the filesystem on a floppy drive as /mnt/floppy, and the filesystem on a CD-ROM as /media/cdrom1, for example. You may also mount files from other systems using a networked filesystem such as NFS. There are other types of file mounts, but this gives you an idea of the process. While the mount process actually mounts the *filesystem* on some device, it is common to simply say that you "mount the device," which is understood to mean "mount the filesystem on the device."

Now, suppose you have just mounted the root file system (/) and you want to mount an IDE CD-ROM, /dev/hdd, at the mount point /media/cdrom. The mount point must exist before you mount the CD-ROM over it. When you mount the CD-ROM, the files and subdirectories on the CD-ROM become the files and subdirectories in and below /media/cdrom. Any files or subdirectories that were already in /media/cdrom are no longer visible, although they still exist on the block device that contained the mount point /media/cdrom. If the CD-ROM is unmounted, then the original files and subdirectories become visible again. You should avoid this problem by not placing other files in a directory intended for use as a mount point.

Table 1 shows the shows the directories required in / by the Filesystem Hierarchy Standard (for more detail on FHS, see Resources).

Table 3. FHS directories in /

Directory	Description
bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a filesystem temporarily
opt	Add-on application software packages
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data

Partitions

The first tutorial in this series, "LPI exam 101 prep (topic 101): Hardware and architecture" touched on partitions on a hard disk, and now we'll go into more detail.

The first IDE hard drive on a Linux system is /dev/hda, and the first SCSI drive is /dev/sda. A hard drive is formatted into 512 byte *sectors*. All the sectors on a disk platter that can be read without moving the head constitute a *track*. Disks usually have more than one platter. The collection of tracks on the various platters that can be read without moving the head is called a *cylinder*. The *geometry* of a hard drive is expressed in cylinders, tracks (or *heads)* per cylinder and sectors/track.

Limitations on the possible sizes of each of these values used with DOS operating systems on PC systems resulted in BIOS translating geometry values so that larger hard drives could be supported. Eventually, even these methods were insufficient. More recent developments in disk drive technology have led to *logical block addressing (LBA)*, so the CHS geometry measurements are less important and the reported geometry on a disk may bear little or no relation to the actual layout of a modern disk. The larger disks in use today have forced an extension to LBA known as LBA48, which reserves up to 48 bits for sector numbers.

The space on a hard drive is divided (or partitioned) into *partitions*. Partitions cannot overlap; space that is not allocated to a partition is called *free space*. The partitions have names like /dev/hda1, /dev/hda2, /dev/hda3, /dev/sda1, and so on. IDE drives are limited to 63 partitions, while SCSI drives are limited to 15. Partitions are usually allocated as an integral number of cylinders (based on the possibly inaccurate notion of a cylinder).

If two different partitioning programs have different understandings of the nominal disk geometry, it is possible for one partitioning program to report an error or possible problem with partitions created by another partitioning program. You may also see this kind of problem if a disk is moved from one system to another, particularly if the BIOS capabilities are different. You can see the nominal geometry on a Linux system by looking at the appropriate geometry special file in the /proc filesystem, such as /proc/ide/hda/geometry. This is the geometry used by partitioning tools like fdisk and parted. Listing 1 shows the use of the `cat` command to display /proc/ide/hda/geometry, followed by the informational messages produced by use of the `parted` partitioning tool.

Listing 1. Hard disk geometry

```
[root@lyrebird root]# cat /proc/ide/hda/geometry
physical     19457/255/63
logical      19457/255/63
[root@lyrebird root]# parted /dev/hda
GNU Parted 1.6.3
Copyright (C) 1998, 1999, 2000, 2001, 2002 Free Software Foundation, Inc.
This program is free software, covered by the GNU General Public License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.  See the GNU General Public License for more details.

Using /dev/hda
Information: The operating system thinks the geometry on /dev/hda is
19457/255/63.  Therefore, cylinder 1024 ends at 8032.499M.
```

```
(parted)
```

Note that, in Listing 1, `parted` has computed a nominal position for the end of cylinder 1024. Cylinder 1024 is important in some older systems where the BIOS is only able to boot partitions that are completely located within the first 1024 cylinders of a disk. This is most likely to occur in a BIOS that does not have LBA support. It is not usually a problem in modern machines, although you should be aware that the limit may exist.

There are three types of partition: *primary*, *logical*, and *extended*. The *partition table* is located in the *master boot record (MBR)* of a disk. The MBR is the first sector on the disk, so the partition table is not a very large part of it. This limits the number of primary partitions on a disk to four. When more than four partitions are required, as is often the case, one of the primary partitions must instead become an extended partition. A disk may contain only one extended partition.

An *extended partition* is nothing more than a container for logical partitions. This partitioning scheme was originally used with MS DOS and PC DOS and permits PC disks to be used by DOS, Windows, or Linux systems.

Linux numbers primary or extended partitions as 1 through 4, so dev hda may have four primary partitions, /dev/hda1, /dev/hda2, /dev/hda3, and /dev/hda4. Or it may have a single primary partition /dev/hda1 and an extended partition /dev/hda2. If logical partitions are defined, they are numbered starting at 5, so the first logical partition on /dev/hda will be/dev/hda5, even if there are no primary partitions and one extended partition (/dev/hda1) on the disk.

Listing 2 shows the output from the `parted` subcommand `p`, which displays the partition information for the disk of Listing 1. Note that this system has several different Windows and Linux filesystems on it.

Listing 2. Displaying the partition table with parted

```
(parted) p
Disk geometry for /dev/hda: 0.000-152627.835 megabytes
Disk label type: msdos
Minor    Start       End     Type      Filesystem  Flags
1          0.031  16300.327  primary   ntfs        boot
2      16300.327  25846.765  primary   fat32       lba
3      25846.765  26842.983  primary   ext3
4      26842.983 152625.344  extended              lba
5      26843.014  28898.173  logical   linux-swap
6      28898.205  48900.981  logical   ext3
7      48901.012  59655.432  logical   ext3
8      59655.463  75657.678  logical   ext3
9      75657.709  95001.569  logical   ext3        boot
10     95001.601 122997.656  logical   reiserfs
11    122997.687 152625.344  logical   ext3
```

Allocating disk space

As mentioned earlier, a Linux system filesystem is a single large tree rooted at /. It is

fairly obvious why data on floppy disks or CD-ROMs must be mounted, but perhaps less obvious why you should consider separating data that is stored on hard drives. Some good reasons for separating filesystems include:

- Boot files. Some files must be accessible to the BIOS or boot loader at boot time.
- Multiple hard drives. Typically each hard drive will be divided into one or more partitions, each with a filesystem that must be mounted somewhere in the filesystem tree.
- Shareable files. Several system images may share static files such as executable program files. Dynamic files such as user home directories or mail spool files may also be shared, so that users can log in to any one of several machines on a network and still use the same home directory and mail system.
- Potential overflow. If a filesystem might fill to 100 percent of its capacity, it is usually a good idea to separate this from files that are needed to run the system.
- Quotas. Quotas limit the amount of space that users or groups can take on a filesystem.
- Read-only mounting. Before the advent of journaling filesystems, recovery of a filesystem after a system crash often took a long time. Therefore, filesystems that seldom changed (such as a directory of executable programs) could be mounted read-only so as not to waste time for checking it after a system crash.

In addition to the filesystem use covered so far, you also need to consider allocating swap space on disk. For a Linux system, this is usually one, or possibly multiple, dedicated partitions.

Making choices

Let's assume you are setting up a system that has at least one hard drive, and you want to boot from the hard drive. (This tutorial does not cover setup for a diskless workstation that is booted over a LAN or considerations for using a live CD or DVD Linux system.) Although it may be possible to change partition sizes later, this usually takes some effort, so making good choices up front is important. Let's get started.

Your first consideration is to ensure that your system will be bootable. Some older systems have a limitation that the BIOS can boot only from a partition that is wholly located within the first 1024 cylinders of disk. If you have such a system, then you **must** create a partition that will eventually be mounted as /boot that will hold the key files needed to boot the system. Once these have been loaded, the Linux system will take over operation of the disk and the 1024 cylinder limit will not affect further operation of the system. If you need to create a partition for /boot, approximately 100

megabytes (MB) is usually sufficient.

Your next consideration is likely to be the amount of required swap space. With current memory prices, swap space represents a very slow secondary memory. A once common rule of thumb was to create swap space equivalent to the amount of real RAM. Today, you might consider allocating approximately 500 MB for a workstation and perhaps 1GB for a server. If special circumstances dictate, you may need to increase these values, but if you do, your system will likely not be performing very well, and you should add real memory. It is possible to use a swap file, but a dedicated partition performs better.

Now we come to a point of divergence. Use of a personal workstation tends to be much less predictable than use of a server. My preference, particularly for new users, is to allocate most of the standard directories (/usr, /opt, /var, etc.) into a single large partition. This is especially useful for new users who may not have a clear idea of what will be installed down the line. A workstation running a graphical desktop and a reasonable number of development tools will likely require 2 or 3 gigabytes of disk space plus space for user needs. Some larger development tools may require several gigabytes each. I usually allocate somewhere between 10 GB and 20 GB per operating system, and I leave the rest of my disk free to load other distributions.

Server workloads will be more stable, and running out of space in a particular filesystem is likely to be more catastrophic. So, for them, you will generally create multiple partitions, spread across multiple disks, possibly using hardware or software RAID or logical volume groups.

You will also want to consider the workload on a particular filesystem and whether the filesystem is shared among several systems or used by just one system. You may use a combination of experience, capacity planning tools, and estimated growth to determine the best allocations for your system.

Regardless of whether you are configuring a workstation or a server, you will have certain files that are unique for each system located on the local drive. Typically, these include /etc for system parameters, /boot for files needed during boot, /sbin for files needed for booting or system recovery, /root for the root user's home directory, /var/lock for lock files, /var/run for running system information, and /var/log for log files for this system. Other filesystems, such as /home for user home directories, /usr, /opt, /var/mail, or /var/spool/news may be on separate partitions, or network mounted, according to your installation needs and preferences.

Section 3. Boot managers

This section covers material for topic 1.102.2 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

This section discusses the PC boot process and the two main boot loaders used in Linux: LILO and GRUB. We will cover choosing a boot manager and recovering when things go wrong.

Boot process overview

Before we get into LILO and GRUB, let's review how a PC starts or *boots*. Code, called *BIOS* (for *B*asic *I*nput *O*utput *S*ervice) is stored in non-volatile memory such as a ROM, EEPROM, or flash memory. When the PC is turned on or rebooted, this code is executed. Usually it performs a power-on self test (POST) to check the machine. Finally, it loads the first sector from the master boot record (MBR) on the boot drive.

As discussed in the previous section on Partitions, the MBR also contains the partition table, so the amount of executable code in the MBR is less than 512 bytes, which is not very much code. Note that every disk, even a floppy, contains executable code in its MBR, even if the code is only enough to put out a message such as "Non-bootable disk in drive A:". This code that is loaded by BIOS from this first sector is called the *first stage boot loader* or the *stage 1 boot loader*.

The standard hard drive MBR used by MS DOS, PC DOS, and Windows operating systems checks the partition table to find a primary partition on the boot drive that is marked as *active*, loads the first sector from that partition, and passes control to the beginning of the loaded code. This new piece of code is also known as the *partition boot record*. The partition boot record is actually another stage 1 boot loader, but this one has just enough intelligence to load a set of blocks from the partition. This new code is called the *stage 2 boot loader*. As used by MS-DOS and PC-DOS, the stage 2 loader proceedes directly to load the rest of operating system. This is how your operating system pulls itself up by the bootstraps until it is up and running.

This works fine for a system with a single operating system. What happens if you want multiple operating systems, say Windows 98, Windows XP, and three different Linux distributions? You **could** use some program (such as the DOS FDISK program) to change the active partition and reboot. This is cumbersome. Furthermore, a disk can have only four primary partitions, and the standard MBR can boot only a primary partition. But our hypothetical example cited five operating systems, each of which needs a partition. Oops!

The solution lies in using some special code that allows a user to choose which operating system to boot. Examples include:

1. Loadlin, a DOS executable program that is invoked from a running DOS system to boot a Linux partition. This was popular when setting up a multi-boot system was a complex and risky process.

2. OS/2 Boot Manager, a program that is installed in a small dedicated partition. The partition was marked active and the standard MBR boot process started Boot Manager, which presented a menu allowing a user to choose which operating system to boot.

3. A smart boot loader, a program that can reside on an operating system partition and is invoked either by the partition boot record of an active partition or by the master boot record. Examples include:
   - BootMagic™, part of Norton PartitionMagic™
   - LILO, the LInux LOader
   - GRUB, the GRand Unified Boot loader

Evidently, if you can get control of the system into a program that has more than 512 bytes of code to accomplish its task, then it isn't too hard to allow booting from logical partitions, or booting from partitions that are not on the boot drive. All of these solutions allow these possibilities, either because they can load a boot record from an arbitrary partition, or because they have some understanding of what file or files to load to start the boot process.

From here on, we will focus on LILO and GRUB as these are the boot loaders included with most Linux distributions. The installation process for your distribution will probably give you a choice of which one to set up. Either will work with most modern disks. Remember that disk technology has advanced rapidly, so you should always make sure that your chosen boot loader, as well as your chosen Linux distribution (or other operating system), as well as your system BIOS will work with your shiny new disk. Failure to do so may result in loss of data.

The stage 2 loaders used in LILO and GRUB allow you to choose which among several operating systems or versions to load. However, LILO and GRUB differ significantly in that a change to the system requires you to use a command to recreate the LILO boot setup whenever you upgrade a kernel or make certain other changes to your system, while GRUB can accomplish this through a configuration text file that you can edit. LILO has been around for a while. GRUB is newer. The original GRUB has now become *GRUB Legacy* and GRUB 2 is being developed under the auspices of the Free Software Foundation (see Resources).

LILO

LILO, or the LInux LOader, is one of the two most common Linux boot loaders. LILO can be installed into the MBR of your bootable hard drive, or into the partition boot record of a partition. It can also be installed on removable devices such as floppy disks, CDs or USB keys. It is a good idea to practice on a floppy disk or USB key if you are not already familiar with LILO, so that is what we will do in our examples.

During Linux installation you will usually specify either LILO or GRUB as a boot manager. If you chose GRUB, then you may not have LILO installed. If you do not, then you will need to install the package for it. We will assume that you already have the LILO package installed. See the package management sections later in this tutorial if you need help with this.

The primary function of `lilo` command, located in /sbin/lilo, is to write a stage 1 boot record and create a map file (/boot/map) using configuration information that is normally located in /etc/lilo.conf. There are some auxiliary uses that we will mention later. First let us look at a typical LILO configuration file that might be used on a dual-boot system with Windows and Linux.

Listing 3. /etc/lilo.conf example

```
prompt
timeout=50
compact
default=linux
boot=/dev/fd0
map=/boot/map
install=/boot/boot.b
message=/boot/message
lba32
password=mypassword
restricted

image=/boot/vmlinuz-2.4.21-32.0.1.EL
        label=linux
        initrd=/boot/initrd-2.4.21-32.0.1.EL.img
        read-only
        append="hdd=ide-scsi root=LABEL=RHEL3"

other=/dev/hda1
        loader=/boot/chain.b
        label=WIN-XP
```

The first set of options above are global options that control how LILO operates. The second and third give per image options for the two operating systems that we want to allow LILO to boot, Red Hat Enterprise Linux 3 or Windows XP in this example.

The global options in our example are:

prompt
forces a boot prompt to be displayed.

timeout
specifies, in tenths of a second, the timeout before the default system will be automatically loaded. Our example of timeout=50 is equivalent to a 5 second timeout.

compact
attempts to merge read requests for adjacent sectors. This speeds up load time and keeps the map smaller.

default
specifies which operating system should be loaded by default. If not specified, the first one is used. IN our example, the Linux system will be loaded if the user does not elect otherwise within the 5 second timeout.

boot
specifies where LILO will be installed. In our example, it is the floppy disk, /dev/fd0. To install in the MBR of the first hard drive specify boot=/dev/hda. Our RHEL 3 system is actually located on /dev/hda11, so we would specify

boot=/dev/hda11 is we wanted to install LILO in this partition. If this parameter is omitted, LILO will attempt to use the boot sector of the device currently mounted as root (/).

map
specifies the location of the map file that LILO uses to provide user prompts and to load the operating systems specified in the per image sections of lilo.conf. By default this is /boot/map.

install
specifies the new file to install as the boot sector. The default is /boot/boot.b, which is provided as part of the LILO package.

message
specifies a message that is displayed before the boot prompt. This must be less than 65535 bytes in length. If your system displays a graphical background with a LILO menu, you may find that /boot/message contains an image file. On some Red Hat systems this will be a 300x200 pixel file in PCX format. On SUSE systems you may find this replaced by a 16 color 640x480 pixel VGA bitmap. In this case, you would also find some additional parameters. Check the documentation that comes with your system. For example, my SUSE SLES9 system has this in /usr/share/doc/packages/lilo/README.bitmaps.

lba32
specifies that LILO should use LBA32 mode for the disk rather than CHS or linear sector addressing.

password
specifies a password that must be entered before booting an image. Note that this is in clear text, so the /etc/lilo.conf file attributes should permit the file to be viewed only by the root user. It should not be the same as your root password. Both password and the next option, restricted, are actually examples of per image options that may be specified in the global section for convenience. If so specified, the same values apply to all images unless overridden in an individual image section.

restricted
relaxes the password requirement so that a password is only required if a user tries to provide additional parameters during boot. You might use this to allow a user to boot normally without entering a password but have to provide a password to boot into single user mode.

The next section gives the per image options for our RHEL3 Linux system.
image
specifies that this section is for a Linux system that should be loaded from a file. The parameter is the filename of a Linux kernel image.

label
is an optional label that can you can enter instead of the full image name to

select this image.

initrd
is the name of the *initial RAM disk* which contains modules needed by the kernel before your file systems are mounted.

read-only
specifies that the root file system should initially be mounted read-only. Later stages of boot will usually remount it read-rwite after it has been checked.

append
specifies options to be passed to the kernel. Our example specifies that SCSI emulation should be used for /dev/hdd (2.4, and earlier, kernels handled optical devices such as CD-ROMs this way). It also specifies that the partition with label RHEL3 should be mounted as root (/).

The final section gives the per image options for our non-Linux system.
other
specifies the name of the device containing the device (or file) that contains the boot sector of the system to be loaded..

loader
specifies the loader to be used. LILO supports chain.b which simply loads that partition boot record from a partition a bootable partition and a variant, /boot/os2_d.b which can be used to boot OS/2 from a second hard drive.

label
is an optional label that can you can enter instead of the full image name to select this image.

Now, if we insert a blank floppy disk we can run the `lilo` command (/sbin/lilo) to create a bootable floppy disk as shown in Listing 4. Note that the lilo command has five levels of verbosity. Specify an extra -v for each level.

Listing 4. Creating a bootable floppy disk with lilo

```
[root@lyrebird root]# lilo -v -v
LILO version 21.4-4, Copyright (C) 1992-1998 Werner Almesberger
'lba32' extensions Copyright (C) 1999,2000 John Coffman

Reading boot sector from /dev/fd0
Merging with /boot/boot.b
Secondary loader: 11 sectors.
Mapping message file /boot/message
Compaction removed 43 BIOS calls.
Message: 74 sectors.
Boot image: /boot/vmlinuz-2.4.21-32.0.1.EL
Setup length is 10 sectors.
Compaction removed 2381 BIOS calls.
Mapped 2645 sectors.
Mapping RAM disk /boot/initrd-2.4.21-32.0.1.EL.img
Compaction removed 318 BIOS calls.
RAM disk: 354 sectors.
Added linux *
```

```
Boot other: /dev/hda1, on /dev/hda, loader /boot/chain.b
Compaction removed 0 BIOS calls.
Mapped 6 (4+1+1) sectors.
Added WIN-XP
/boot/boot.0200 exists - no backup copy made.
Map file size: 8192 bytes.
Writing boot sector.
```

We now have out bootable LILO diskette. If LILO encounters an error, you might see an error message and the boot sector will not be written. For example, if we had omitted the lba32 option from our /etc/lilo.conf file we might see output such as that in Listing 5. This would be a tip to try the linear or lba32 options. In this case, we use the command line to specify the -l option which is equivalent to specifying the linear option in lilo.conf. If we were to do this again with the -L option, lilo should be successful and the output should be as in the previous listing.

Listing 5. Incorrect /etc/lilo.conf example

```
[root@lyrebird root]# lilo
Warning: device 0x030b exceeds 1024 cylinder limit
Fatal: geo_comp_addr: Cylinder number is too big (16284 > 1023)
[root@lyrebird root]# lilo -l
Warning: device 0x030b exceeds 1024 cylinder limit
Fatal: sector 261613688 too large for linear mode (try 'lba32' instead)
```

When you have tested your boot diskette, change the boot=/dev/fd0 entry in your lilo.conf file to install LILO on the MBR or a partition boot record. For example, boot=/dev/hda will install LILO in the master boot record of your first IDE hard drive.

You now have an introduction to LILO and its configuration file, including how to override some configuration options from the lilo command line. You will find more information in the lilo man page using the command `man lilo`. You will find even more extensive information in the postscript user guide that is installed with the lilo package. This should be installed in your documentation directory, but the exact location may vary by system. One way to locate the file is to filter the package list through grep. Listing 6 shows this for the rpm-based RHEL3 system that we have been using in this example.

Listing 6. Locating the LILO user guide with rpm.

```
[ian@lyrebird ian]$ rpm -ql lilo | grep ".ps$"
/usr/share/doc/lilo-21.4.4/doc/Technical_Guide.ps
/usr/share/doc/lilo-21.4.4/doc/User_Guide.ps
```

LILO auxiliary commands

LILO has several auxiliary commands.

lilo -q
: will display information from the map file

lilo -R
: will set lilo to automatically boot the specified system on the next reboot only. This is very convenient for automatically rebooting remote systems.

lilo -I
will display information about the path of a kernel

lilo -u
will uninstall lilo and restore the previous boot record.

When LILO boots a Linux system you may want to provide additional parameters at boot time. For example, if your graphical startup was not working, you may want to boot into mode 3 or into single user mode to recover. Any text you type after the label name will be passed to the kernel. For example, in our example, we would select the RHEL system by simply typing "linux". To boot into mode 3 or single user mode we would type one of the following strings as appropriate.

```
linux 3
linux single
```

Remember also that with LILO you **must** run the lilo command whenever you update the configuration file (/etc/lilo.conf). You should also run the lilo command if you add, move, or remove partitions or make any other changes that might invalidate the generated boot loader.

GRUB

GRUB, or the GRand Unifood Boot loader, is the other of the two most common Linux boot loaders. As with LILO, GRUB can be installed into the MBR of your bootable hard drive, or into the partition boot record of a partition. It can also be installed on removable devices such as floppy disks, CDs or USB keys. It is a good idea to practice on a floppy disk or USB key if you are not already familiar with GRUB, so that is what we will do in our examples.

GRUB, or GNU GRUB, is now developed under the auspices of the Free Software Foundation. A new version, GRUB 2 is under development, so the original GRUB 0.9x versions are now known as Grub Legacy.

During Linux installation you will usually specify either LILO or GRUB as a boot manager. If you chose LILO, then you may not have GRUB installed. If you do not, then you will need to install the package for it. We will assume that you already have the GRUB package installed. See the package management sections later in this tutorial if you need help with this.

GRUB has a configuration file which is usually stored in /boot/grub/grub.conf. If your filesystem supports symbolic links, as most Linux filesystems do, you will probably have /boot/grub/menu.lst as a symbolic link to /boot/grub/grub.conf.

The `grub` command (/sbin/grub, or, on some systems, /usr/sbin/grub) is a small, but reasonably powerful shell which supports several commands for installing GRUB, booting systems, locating and displaying configuration files and similar tasks. This shell shares much code with the second stage GRUB boot loader, so it is useful to

learn about GRUB without having to boot to a second stage GRUB environment. The GRUB stage 2 runs either in menu mode or command mode, to allow you to choose an operating system from a menu, or to specify individual commands to load a system. There are also several other commands, such as `grub-install` which use the grub shell and help automate tasks such as installing GRUB.

Listing 7 shows part of a GRUB configuration file. As you look through it, remember one important thing -- GRUB counting for drives, partitions and things that need to be counted starts at 0 rather than 1.

Listing 7. /boot/grub/menu.lst GRUB configuration example.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /, eg.
#          root (hd1,5)
#          kernel /boot/vmlinuz-version ro root=/dev/hdc6
#          initrd /boot/initrd-version.img
#boot=/dev/hdc6
default=2
timeout=10
splashimage=(hd0,6)/boot/grub/splash.xpm.gz
password --md5 $1$/8Kl21$3VPIphs6REHeHccwzjQYO.

title Red Hat Linux (2.4.20-31.9)
        root (hd0,6)
        kernel /boot/vmlinuz-2.4.20-31.9 ro root=LABEL=RH9 hdd=ide-scsi
        initrd /boot/initrd-2.4.20-31.9.img

title Red Hat Linux (2.4.20-6)
        root (hd0,6)
        kernel /boot/vmlinuz-2.4.20-6 ro root=LABEL=RH9 hdd=ide-scsi
        initrd /boot/initrd-2.4.20-6.img

title Red Hat Enterprise Linux WS A (2.4.21-32.0.1.EL)
        root (hd0,10)
        kernel /boot/vmlinuz-2.4.21-32.0.1.EL ro root=LABEL=RHEL3 hdd=ide-scsi
        initrd /boot/initrd-2.4.21-32.0.1.EL.img

title         Ubuntu, kernel 2.6.10-5-386
root         (hd1,10)
kernel         /boot/vmlinuz-2.6.10-5-386 root=/dev/hdb11 ro quiet splash
initrd         /boot/initrd.img-2.6.10-5-386
savedefault
boot

title         Ubuntu, kernel 2.6.10-5-386 (recovery mode)
lock
root         (hd1,10)
kernel         /boot/vmlinuz-2.6.10-5-386 root=/dev/hdb11 ro single
initrd         /boot/initrd.img-2.6.10-5-386
boot

title Win/XP
        rootnoverify (hd0,0)
        chainloader +1

title Floppy
        root (fd0)
        chainloader +1
```

As with the LILO configuration file, the first set of options above control how GRUB operates. For GRUB, these are called *menu commands* and they must appear before other commands. The remaining sections give per image options for the

operating systems that we want to allow GRUB to boot. Note that "title" is considered a menu command. Each instance of title is followed by one or more general or menu entry commands. Our LILO example was a typical basic example for a dual boot system with Windows and Linux. This example comes from the same system as we used before, but here we have added a few extra operating systems, to show you some of the power of a boot loader. You will recognize many of the same kinds of elements occurring in both LILO and GRUB configuration files. You might like to think about what would need to change if you added the extra operating systems here to the earlier LILO example.

The menu commands that apply to all other sections in our example are:

#

Any line starting with a # is a comment and is ignored by GRUB. This particular configuration file was originally generated by anaconda, the Red Hat installer. You will probably find comments added to your GRUB configuration file if you install GRUB when you install Linux. The comments will often serve as an aid to the system upgrade program so that your GRUB configuration can be kept current with upgraded kernels. Pay attention to any markers that are left for this purpose if you edit the configuration yourself.

default

specifies which system to load if the user does not make a choice within a timeout. In our example, default=2 means to load the **third** entry. Remember that GRUB counts from 0 rather than 1. If not specified, then the default is to boot the first entry, entry number 0.

timeout

specifies a timeout in seconds before booting the default entry. Note that LILO uses tenths of a second for timeouts while GRUB uses whole seconds.

splashimage

Specifies the background, or *splash*, image to be displayed with the boot menu. GRUB refers to the first hard drive as (hd0) and the first partition on that drive as (hd0,0), so the specification of splashimage=(hd0,6)/boot/grub/splash.xpm.gz means to use the file /boot/grub/splash.xpm.gz located on partition 7 of the first hard drive. Remember that bit about counting from 0. Note also, that the image is an XPM file compressed with gzip. Support for splashimage is a patch that may or may not be included in your distribution.

password

specifies a password that must be entered before a user can unlock the menu and either edit a configuration line or enter GRUB commands. As with LILO, the password may be in clear text. GRUB also permits passwords to be stored as an MD5 digest, as in our example. This is somewhat more secure and most administrators will set a password. Without a password, a user has complete access to the GRUB command line.

Our example shows five Linux distributions (three Red Hat and two Ubuntu) plus a

Windows XP and a floppy boot option. The commands used in these sections are:

title
is a descriptive title that is shown as the menu item when Grub boots. You use the arrow keys to move up and down through the title list and then press the **Enter** key to select a particular entry.

root
specifies the partition that will be booted. As with splashimage, remember that counting starts at 0, so the first Red Hat system which is specified as root (hd0,6) is actually on partition 7 the first hard drive (/dev/hda7 in this case), while the first Ubuntu system which is specified as root (hd1,10) is on the second hard drive (/dev/hdb11). GRUB will attempt to mount this partition to check it and provide values to the booted operating system in some cases.

kernel
specifies the kernel image to be loaded and any required kernel parameters. This is similar to a combination of the LILO image and append commands. We have two different Red Hat 9 kernels, plus a Red Hat Enterprise Linux 3 Workstation kernel, and one level of Ubuntu system with two different sets of kernel parameters in this example.

initrd
is the name of the *initial RAM disk* which contains modules needed by the kernel before your file systems are mounted.

savedefault
is shown here for illustration. If the menu command `default=saved` is specified, and the savedefault command is specified for an operating system, then booting that operating system will cause it to become the default until another operating system with savedefault specified is booted. In this example, the specification of default=2 will override any saved default.

boot
is an optional parameter that instructs GRUB to boot the selected operating system. This is the default action when all commands for a selection have been processed.

lock
is used in this example with the second Ubuntu system. This system will boot into single user mode which permits a user to make modifications to a system that are normally restricted to root access. If this is specified, then you should also specify a password in the initial options, otherwise, a user can edit out your lock option and boot the system, or add "single" to one of the other entries. It is also possible to specify a different password for individual entries if you wish.

rootnoverify
is similar to root, except that GRUB does not attempt to mount the filesystem or verify its parameters. This is usually used for filesystems such as NTFS that are not supported by GRUB. You might also use this if you wanted GRUB to

load the master boot record on a hard drive, for example to access a different configuration file, or to reload your previous boot loader.

chainloader
specifies that another file will be loaded as a stage 1 file. The value "+1" is equivalent to 0+1 which means to load one sector starting at sector 0, that is, load the first sector from the device specified by root or rootnoverify.

You now have some idea of what you might find in a typical /boot/grub/grub.conf (or /boot/grub/menu.lst) file. There are many many other GRUB commands to provide extensive control over the boot process as well as help with installing grub and other tasks. You can learn more about these in the GRUB manual which should be available on your system using the command `info grub`.

Now that we have a GRUB configuration file, we need to create a boot floppy to test it. The simplest way to do this is to use the `grub-install` command as shown in Listing 8. If you are installing GRUB onto a floppy or onto a partition, then you should unmount the device first. This does not apply if you are installing GRUB in the MBR of a hard drive, as you only mount partitions (/dev/hda1, /dev/hda2, etc.) and not the whole hard drive (/dev/hda).

Listing 8. Installing GRUB to a floppy disk.

```
[root@lyrebird root]# umount /dev/fd0
umount: /dev/fd0: not mounted
[root@lyrebird root]# grub-install /dev/fd0
Installation finished. No error reported.
This is the contents of the device map /boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(fd0)   /dev/fd0
(hd0)   /dev/hda
(hd1)   /dev/hdc
(hd2)   /dev/sda
```

Note: You may also use the GRUB device name (fd0) instead of /dev/fd0, but if you do you must enclose it in quotes to avoid shell interpretation of the parentheses. For example:

```
grub-install '(fd0)'
```

If you started with an empty floppy, and now now mount it, you will discover that it still appears to be empty. What has happened is that GRUB wrote a customized stage 1 loader to the first sector of the disk. This does not show up in the file system. This stage 1 loader will load stage 2 and the configuration file from your hard drive. Try booting the diskette and you will see very little IO activity before your menu is displayed.

The device map tells you how GRUB will match its internal view of your disks (fd0, hd0, hd1) to the Linux view (/dev/fd0, /dev/hda, /dev/hdb). On a system with one or two IDE hard drives and perhaps a floppy drive, this will probably be correct. If a

device map already exists, GRUB will reuse it without probing. If you just added a new drive and want to force a new device map to be generated add the `--resize` option to the grub-install command. For example

Once you have tested your boot floppy you are ready to install GRUB in the MBR of your hard drive. For the first IDE hard drive, you would use:

```
grub-install /dev/hda
or
grub-install '(hd0)'
```

To install it into the partition boot record for partition 11, use:

```
grub-install /dev/hda11
or
grub-install '(hd0,10)'
```

Remember that GRUB numbers from 0.

System updates

Most distributions provide tools for updating the system. These tools are usually aware of the boot loader in use and will often update your configuration file automatically. If you build your own custom kernel, or prefer to use a configuration file with a non-standard name or location, then you may need to update the configuration file yourself.

- If you use LILO, then you **must** run the lilo command whenever you update your configuration file or make changes such as adding a hard drive or deleting a partition.
- If you use GRUB, you can edit the /boot/grub/grub.conf file to make your changes and the GRUB stage 2 loader will read the file when you reboot. You do not normally need to reinstall GRUB just because you add a new kernel. However, if you move a partition, or add drives, you may need to reinstall GRUB. Remember the stage 1 loader is very small, so it simply has a list of block addresses for the stage 2 loader. Move the partition and the addressed change, so stage 1 can no longer locate stage 2. We'll cover some recovery strategies and also discuss GRUB's stage 1.5 loaders next.

Recovery

We will now look at some things that can go wrong with your carefully prepared boot setup, particularly when you install and boot multiple operating systems.The first thing to remember is to resist your initial temptation to panic. Recovery is usually only a few steps away. We will give you a few strategies here that should help you through many types of crisis.

These strategies and tools will show you that anyone who has physical access to a machine has a lot of power. Likewise, anyone who has access to a grub command line also has access to files on your system without the benefit of any ownership or other security provisions provided by a running system. Keep these points in mind when you select your boot loader. The choice between LILO and GRUB is largely a matter of personal preference. with what you have learned already and what we are about to show you, you should be equipped to choose the loader that best suits your particular needs and style of working.

Another install destroys your MBR.

Sometimes you will install another operating system and inadvertently overwrite your MBR. Some systems, such as DOS and Windows, always install their own MBR. It is usually very easy to recover from this situation. If you develop a habit of creating a boot floppy every time you run lilo or reinstall GRUB, you are home free. Simply boot into your Linux system from the floppy and rerun lilo or grub-install.

If you don't happen to have a boot floppy, but you still have almost any Linux distribution available, you can usually boot the Linux install media in a recovery mode. When you do so, the root filesystem on your hard drive will either be mounted at some strange recovery point, or the drive will not be mounted at all. You can use the `chroot` command to make this odd mount point become your root (/) directory. Then run lilo or grub-install to create a new boot floppy or to reinstall the MBR. I prefer to create a floppy and use it to boot, making sure that all is well before I go and rewrite the MBR, but you may be more courageous than I. Listing 9 shows an example using the environment we used for our earlier configuration examples. In this example, I booted a Red Hat Enterprise Linux boot disk which mounted /dev/hda11 at /mnt/sysimage. Most rescue environments will dump you into a large screen with a prompt, rather than the graphical screen you might be more used to. Think of this as a terminal window with you logged in as root. In other words, be very careful what you write to your hard drive. In listing 9, user entry is shown in**bold**.

Listing 9. Using a rescue disk and chroot.

```
sh-3.00# chroot /mnt/sysimage
sh-2.05b# lilo
Added linux *
Added WIN-XP
sh-2.05b# grub-install '(fd0)'
Installation finished. No error reported.
This is the contents of the device map /boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(fd0)   /dev/fd0
(hd0)   /dev/hda
(hd1)   /dev/hdc
(hd2)   /dev/sda
sh-2.05b#
```

Once you have your bootable floppy, press ctrl-d to exit from the chroot environment and then reboot your system, remembering to remove your installation media. If you don't happen to have an installation CD or DVD handy, there are many recovery and live LINUX CDs available online and some diskette or USB memory key ones too.

See Resources.

Although beyond the scope of this tutorial, you may wish to know that it is possible to have your MBR boot a Windows 2000 or Windows XP system and install Lilo or GRUB on a partition boot record. The ntldr boot program can also chain load other boot sectors, although setup can be a little tricky. You will need to copy the boot sector to a Windows partition and modify the hidden boot.ini file to make this work.

You moved a partition.

If you moved a partition and forgot about your boot setup, you have a temporary problem. Typically, LILO or GRUB refuse to load. LILO will probably print an 'L' indicating that stage 1 was loaded and then stop. GRUB will give you an error message. What has happened here is that the stage 1 loader, which had a list of sectors to load to get to the stage 2 loader, can perhaps load the sectors from the addresses it has, but the sectors no longer have the stage 2 signature. If you built a boot diskette using the methods outlined earlier, remember that all that wither lilo or grub-install put on the diskette was a single boot sector, so your boot diskette probably won't help. As in the previous example, you will probably need to boot some kind of rescue environment and rebuild your boot floppy with LILO or GRUB. Then reboot, check your system and reinstall your boot loader in the MBR.

You may have noticed that our configuration examples used labels for partitions. for example,

```
append="hdd=ide-scsi root=LABEL=RHEL3"
or
kernel /boot/vmlinuz-2.4.20-31.9 ro root=LABEL=RH9 hdd=ide-scsi
```

I often use labels like this to help avoid problems when I move partitions. You still need to update the GRUB or LILO configuration file and rerun lilo, but you don't have to update /etc/fstab as well. This is particularly handy if I create a partition image on one system and restore it at a different location on another system.

Using a /boot partition.

Another approach to recovery, or perhaps avoiding it, is to use a separate partition for /boot. This partition need not be very large, perhaps 100MB or so. Put this partition somewhere where it is unlikely to be moved and where it is unlikely to have its partition number moved by the addition or removal of another partition. In a mixed Windows and Linux environment, /dev/hda2 is often a good choice for a partition for /boot.

Another reason for having a /boot partition arises when your root partition uses a file system not supported by your boot loader. For example, it is very common to have a /boot partition formatted ext2 or ext3 when the root partition (/) uses LVM.

If you have multiple distributions on your system, **do not** share the /boot partition between them. Remember to set up LILO or GRUB to boot from the partition that will later be mounted as /boot. Remember also that the update programs for a

distribution will usually update the GRUB or LILO configuration for that system. In an environment with multiple systems you may want to keep one with its own /boot partition as the main one and manually update that configuration file whenever an update of one of your systems requires it. Another approach is to have each system install a boot loader into its own partition boot record and have your main system simply chain load the partition boot records for the individual systems, giving you a two-stage menu process.

Building a self-contained boot diskette.

Finally, let's look a little more at the GRUB setup and how to make a standalone boot diskette that will get you to a GRUB prompt, no matter what has happened to your hard drive.

Remember all that stuff about cylinders on hard drives. Even though you might think of a cylinder as a fictional entity with modern drives, many aspects of your file system have not forgotten them. In particular, you will find partition use an integral number of cylinders aligned on cylinder boundaries. Within a partition, many file systems also manage space in units of cylinders. On many UNIX and Linux systems, the layout of the filesystem is stored in a *superblock* which is the first allocation unit in the file system. For systems such as ext2 or ext3 filesystems and reasonably large hard drives, the space will be broken into several sections with a copy of the superblock in the beginning of each section. This will help with recovery if you accidentally mess up partition boundaries with a program like fdisk.

One other benefit of the cylinder mentality is that there is some space at the beginning of a disk right after the MBR. GRUB takes advantage of this by embedding a stage 1.5 boot loader in this space or in similar otherwise unused space on a partition whenever possible. The stage 1.5 loader understands the file system on the partition that contains the stage 2, so it is somewhat more protected against problems associated with files being moved.

All that is well and good, but how does it relate to a bootable floppy. well, a floppy doesn't have much space or much notion of cylinders, so if you want to boot both stage 1 and stage 2 of GRUB from a floppy, you need to install stage 1 and then copy stage 2 to the sectors immediately following the boot sector. Listing 10 shows an example of how to do this. Use an empty diskette as this process will destroy data. You should copy the files that came with your grub distribution rather than the ones from your /boot/grub directory as /boot/grub/stage2 has been modified to work with your hard drive partitions. You should find the original stage1 and stage2 files in a subdirectory of /usr/share/grub. In our example they are in /usr/share/grub/i386-redhat.

Listing 10. Creating a GRUB boot floppy.

```
[root@lyrebird root]# ls /usr/share/grub
i386-redhat
[root@lyrebird root]# cd /usr/share/grub/i386-redhat
[root@lyrebird i386-redhat]# ls -l st*
-rw-r--r--    1 root     root          512 Aug  3  2004 stage1
-rw-r--r--    1 root     root       104092 Aug  3  2004 stage2
```

```
[root@lyrebird i386-redhat]# dd if=stage1 of=/dev/fd0 bs=512 count=1
1+0 records in
1+0 records out
[root@lyrebird i386-redhat]# dd if=stage2 of=/dev/fd0 bs=512 seek=1

203+1 records in
203+1 records out
```

If you had formatted your floppy before you did this, and you now attempt to mount the floppy, the mount command will give you an error. Copying the stage2 right after the diskette's boot sector (seek=1) destroyed the filesystem on your diskette.

If you now boot this diskette, you will notice that the delay while it loads stage 2 from the diskette. You could boot this diskette in an arbitrary PC; it does not have to be one with a Linux system on it. When you boot the diskette, you will get a GRUB boot prompt. Press the tab key to see a list of commands available to you. Try `help commandname` to get help on the command called *commandname*. Listing 11, illustrates the GRUB command line.

Listing 11. The GRUB command line.

```
    GRUB  version 0.93  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename.]

grub>
 Possible commands are: blocklist boot cat chainloader clear cmp color configfi
le debug device displayapm displaymem dump embed find fstest geometry halt help
 hide impsprobe initrd install ioprobe kernel lock makeactive map md5crypt modu
le modulenounzip pager partnew parttype password pause quit read reboot root ro
otnoverify savedefault serial setkey setup terminal terminfo testload testvbe u
nhide uppermem vbeprobe

grub> help rootnoverify
rootnoverify: rootnoverify [DEVICE [HDBIAS]]
    Similar to `root', but don't attempt to mount the partition. This
    is useful for when an OS is outside of the area of the disk that
    GRUB can read, but setting the correct root device is still
    desired. Note that the items mentioned in `root' which derived
    from attempting the mount will NOT work correctly.

grub> find /boot/grub/grub.conf
 (hd0,2)
 (hd0,6)
 (hd0,7)
 (hd0,10)
 (hd1,7)

grub>
```

In this example, we have found that there are GRUB config files on four different partitions on our first hard drive and another on the second hard drive. We could load the GRUB menu from one of these using the configfile command. For example:

```
configfile (hd0,2)/boot/grub/grub.conf
```

This would load the menu for that configuration file and we might be able to boot the system from this point. You can explore these grub commands in the GRUB manual. Try typing `info grub` in a Linux terminal window to open the manual.

One last point before we leave GRUB. We mentioned that the stage 2 GRUB file destroyed the file system on the diskette. If you want a bootable GRUB recovery diskette that loads GRUB files, including a configuration file from the diskette you can accomplish this using the following steps:

1. Use the `mkdosfs` command to create a DOS FAT filesystem on the diskette and use the `-R` option to reserve enough sectors for the stage 2 file.
2. Mount the diskette
3. Create a /boot/grub directory on the diskette
4. Copy the GRUB stage1, stage2, and grub.conf files to the boot/grub directory on the diskette. Copy your splash image file too if you want one.
5. Edit your grub.conf file on the diskette so it refers to the splash file on the diskette.
6. Unmount the diskette
7. Use the `grub` command shell to setup GRUB on the diskette using the GRUB root and setup commands.

We illustrate this in Listing 12.

Listing 12. Installing GRUB on diskette with a filesystem.

```
[root@lyrebird root]# mkdosfs -R 210 /dev/fd0
mkdosfs 2.8 (28 Feb 2001)
[root@lyrebird root]# mount /dev/fd0 /mnt/floppy
[root@lyrebird root]# mkdir /mnt/floppy/boot
[root@lyrebird root]# mkdir /mnt/floppy/boot/grub
[root@lyrebird root]# cp /boot/grub/stage1 /mnt/floppy/boot/grub
[root@lyrebird root]# cp /boot/grub/stage2 /mnt/floppy/boot/grub
[root@lyrebird root]# cp /boot/grub/splash* /mnt/floppy/boot/grub
[root@lyrebird root]# cp /boot/grub/grub.conf /mnt/floppy/boot/grub
[root@lyrebird root]# umount /dev/fd0
[root@lyrebird root]# grub
Probing devices to guess BIOS drives. This may take a long time.

    GRUB  version 0.93  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename.]

grub> root (fd0)
 Filesystem type is fat, using whole disk

grub> setup (fd0)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/fat_stage1_5" exists... no
 Running "install /boot/grub/stage1 (fd0) /boot/grub/stage2 p /boot/grub/grub.c
onf "... succeeded
Done.
```

With these tools you should now have enough to recover from most of the things

that can go wrong with a boot loader.

Section 4. Make and install programs

This section covers material for topic 1.102.3 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 5.

In this section, you will learn how to build and install an executable program from source. You will learn how to unpack typical source bundles and customize Makefiles.

Why might you want to install a program from source? Frequent reasons include:

1. You need a program that is not part of your distribution.
2. you need a program that is only available as source.
3. You need some feature of a program that is only available by rebuilding the program from source.
4. You want to learn more about how a program works or you want to participate in its development.

These are all good reasons to consider installing a program from source.

Download and unpack

Regardless of your motivation for building from source, you will need to get the source before you can build it. You might find the package on a web site designed for hosting projects, such as the Open Source Technology Group's SourceForge.net (see Resources)., or a web site dedicated to a particular package.

In this section we will look mostly at packages distributed as so-called *tarballs*. The `tar` (for *T*ape *AR*chive) command is used to create and manipulate *archives* from the files in a directory tree. Despite the name, the files can be stored on any media. In fact, storing them on disk allows manipulation, such as deleting part of an archive, that is not even possible on tape. The tar command itself does not compress the data, it merely stores it in a form from which the original files, permissions and directory structures can be restored. The tar command can be used in conjunction with a compression program, normally `gzip` or `bzip2` to create a compressed archive which saves storage space as well as transmission time. Such a compressed archive is a tarball.

Besides simple tarballs, source for a particular program may be packaged for your distribution in a *source package*, such as a source RPM (or SRPM). We will discuss

package management later in this tutorial. For now, just remember to check for a source package for your distribution if one is available as that is usually a slightly easier way to build a program and it will already be tailored to the filesystem layout used by your distribution.

Before you download, try to learn as much as you can about the package. If there is installation or build documentation available, check it to see if you will need other packages in order to build it. Frequently, you will also need to install several libraries and perhaps development tools before you can successfully build your chosen program. This is especially likely to be true if your program uses any graphical toolkit. Sometimes you will start the build process and only then discover that you need a particular package. Don't worry, this is not uncommon. You'll just need to locate and install the missing packages and keep trying till you have all the required packages.

You will usually download using your browser or perhaps an ftp program. Your package will probably have a name that ends in one of tar, tar.gz, tar.Z, tgz, or tar.bz2. Sometimes you will download a package using CVS (Concurrent Version System). An example might be GNU GRUB 2 from the Free Software Foundation (see Resources). In this case, your downloaded source will be unpacked already. Occasionally you may find a .zip extension indicating a zip file.

Compressed tar files

Compressed tar files or *tarballs*) are the most common form of source distribution for source that is not using package management such as Red Hat's RPM, or Debian's package management. These are created using the `tar` command which archives a directory tree and all its files in a single file. This will usually be compressed with some form of compression program, the usual ones being either `compress`, `gzip`, or `bzip2`. Because archiving and compressing is such a common operation, the GNU tar command found on most Linux systems can also handle compression and decompression using compress, gzip or bzip2. If your particular version of tr does not handle the particular compression type, UNIX and Linux systems are very good at using pipelines to allow several commands to operate in sequence on one input source, so a two-stage process can accomplish manually what tar does for you under the covers anyway.

For illustration, suppose we download the Dr. Geo interactive geometry project (see Resources). At the time of writing, we downloaded drgeo-1.1.0.tar.gz. The gz extension tells us that this file is compresses with gzip. We will first show you how to extract the tar file from the compressed file and then how to extract the files from the tar archive. Then we will show you how to uncompress and extract with a single command or a pipeline.

To just extract the tar archive we use the `gunzipg` command as shown in Listing 13.

Listing 13. Decompressing the Dr Geo source package.

```
[ian@localhost ~]$ ls drgeo*
drgeo-1.1.0.tar.gz
[ian@localhost ~]$ gunzip drgeo-1.1.0.tar.gz
[ian@localhost ~]$ ls drgeo*
drgeo-1.1.0.tar
```

Note that our .tar.gz file has now been replace by a plain .tar file. For the other extensions mentioned, you would use one of the following commands (as appropriate) to extract the compressed tar file.

```
uncompress drgeo-1.1.0.tar.Z
gunzip drgeo-1.1.0.tar.Z
gunzip drgeo-1.1.0.tar.gz
gunzip drgeo-1.1.0.tgz
bunzip2 drgeo-1.1.0.tar.bz2
```

You'll notice that gunzip will handle .Z, .tar.gz and .tgz. In fact, your system may not even have the the earlier compress and uncompress programs installed at all.

To extract the files from the tar archive, you use the `tar` command. The usual form, `tar -xvf filename.tar`, is shown in Listing 14. Optionally, you may pipe the output through the less filter in order to page through it.

Listing 14. Extracting files from the Dr Geo archive.

```
[ian@localhost ~]$ tar -xvf drgeo-1.1.0.tar |more
drgeo-1.1.0/
drgeo-1.1.0/po/
drgeo-1.1.0/po/ChangeLog
drgeo-1.1.0/po/Makefile.in.in
drgeo-1.1.0/po/POTFILES.in
drgeo-1.1.0/po/drgeo.pot
drgeo-1.1.0/po/az.po
drgeo-1.1.0/po/ca.po
drgeo-1.1.0/po/cs.po
...
```

The `-x` option tells tar to extract the files. The `-v` option tells tar to give verbose output. And the `-f` option , in conjunction with a file name (drgeo-1.1.0.tar in this case) tells tar what archive file to extract files from.

Well-behaved packages will create a directory in which to store the files of the package. In our example, this is the drgeo-1.1.0 directory. Occasionally, a package will not do this, so you might want to check before dumping a lot of files over your home directory. To do this, use the tar command with the `-t` option to display the table of contents, instead of using the `-x` to do the extraction. If you also drop the `-v` option, you will get enough output to see what files will be created and whether a directory will be created or whether everything will be dumped into the current directory.

Now that we have seen how to extract a tarball in two steps you are perhaps wondering about the claim that it could be done in one. It can. If you add the `-z` option to the tar command it can decompress and extract gzipped archives with a single command. For example:

```
tar -zxvf drgeo-1.1.0.tgz
or
tar -zxvf drgeo-1.1.0.tar.Z
```

To accomplish the same thing with an archive compressed with bzip2, use the `-j` option instead of the `-z` option . For example:

```
tar -jxvf drgeo-1.1.0.tar.bz2
```

You can also use the -c option on any of the above decompression commands to direct the decompressed file to standard output, which you then pipe as standard input the tar command. Note that this will leave your original file unchanged, rather than extracting it to a larger .tar file. Some examples are:

```
bunzip2 -c drgeo-1.1.0.tar.bz2 | tar -xvf -
uncompress -c drgeo-1.1.0.tar.Z | tar -xvf -
gunzip -c drgeo-1.1.0.tar.Z | tar -xvf -
gunzip -c drgeo-1.1.0.tar.gz | tar -xvf -
gunzip -c drgeo-1.1.0.tgz | tar -xvf -
```

Notes:

1. The `-` value for an archive file name tells tar to use standard input for the archive. Your version of tar may do this by default, in which case you do not need to specify the `-f` option at all. Just leave the trailing `f -` off the above commands.

2. The `zcat` command performs the same function as `gunzip -c`.

CVS trees

Sometimes the code for the project you need is not packaged in a tarball but is available through CVS (Concurrent Version System). At the time of writing, an example is the GRUB 2 project that we discussed in the section Boot managers. Listing 15, shows an example.

Listing 15. Downloading GRUB2 with CVS.

```
[ian@attic4 ~]$ export CVS_RSH="ssh"
[ian@attic4 ~]$ cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/grub co grub2
cvs server: Updating grub2
U grub2/.cvsignore
U grub2/AUTHORS
U grub2/COPYING
U grub2/ChangeLog
U grub2/DISTLIST
U grub2/INSTALL
U grub2/Makefile.in
U grub2/NEWS
...
```

The export command tells CVS how to connect to the remote server (using secure shell, or ssh, in this case). The cvs command checks out (co option) the grub2

project. You will find all the project files in the grub2 directory that the cvs command created for you.

Zip files

You will occasionally find source packaged as a zip file. This might be the case for a package which works on Windows as well as Linux or UNIX systems. The original PKZIP program was developed for DOS systems by PKWARE, Inc. and is now available on several other platforms. Many Linux systems include a version created by Info-ZIP.

Listing 16 shows how to use the `unzip` command to extract source for the sphere eversion program and screensaver.

Listing 16. Unzipping the sphere eversion source.

```
[ian@attic4 ~]$ unzip sphereEversion-0.4-src.zip
Archive:  sphereEversion-0.4-src.zip
   creating: sphereEversion-0.4-src/
  inflating: sphereEversion-0.4-src/Camera.h
  inflating: sphereEversion-0.4-src/drawutil2D.h
  inflating: sphereEversion-0.4-src/drawutil.h
  inflating: sphereEversion-0.4-src/fontdata.h
  inflating: sphereEversion-0.4-src/fontDefinition.h
  inflating: sphereEversion-0.4-src/generateGeometry.h
  inflating: sphereEversion-0.4-src/global.h
  inflating: sphereEversion-0.4-src/mathutil.h
  inflating: sphereEversion-0.4-src/Camera.cpp
  inflating: sphereEversion-0.4-src/drawutil2D.cpp
  inflating: sphereEversion-0.4-src/drawutil.cpp
  inflating: sphereEversion-0.4-src/fontdata.cpp
  inflating: sphereEversion-0.4-src/generateGeometry.cpp
  inflating: sphereEversion-0.4-src/main.cpp
  inflating: sphereEversion-0.4-src/mathutil.cpp
  inflating: sphereEversion-0.4-src/README.TXT
  inflating: sphereEversion-0.4-src/Makefile
```

Build ithe program

Now that you have your source files unpacked into a directory tree let's look at how to build the program or programs.

Inspecting the source

Before you start building, you should look over what you unpacked. In particular, look for installation documentation. There will usually be a README or an INSTALL file, or perhaps both, located in the root directory of your new project. If the package is intended for multiple platforms, you might also find platform specific files such as README.linux or INSTALL.linux.

Configuration

You will often find a *configure* script in the main source directory. This script is designed to set up a *Makefile* that is customized to your system. It is often generated by the developers using the GNU autoconf program. The configure script will probe

your system to determine its capabilities. The resulting Makefile, or Makefiles, will build the project on your particular system.

A complex configuration script may check many aspects of your system, including things such as processor type, whether it is a 32-bit or 64-bit system and so on. A simple configuration script may do little more than create Makefiles.

If you don't have a file called *configure* in your main project directory, check your documentation to see if there is some alternate method. If you do, try changing to the main project directory and running

```
./configure --help
```

This should give you some help on available configuration options. Many, such as --prefix, will occur in most configure scripts. Some are likely to be specific to the particular program you are building. Note the ones that you'd like to change.

Note: If your project does not include a configuration script, then it will probably have a Makefile which will work on most platforms, or some other form of installation process. For example, a package that uses only Python scripts and data files may not need to be built, so it may just have a script for installation.

In the tutorial for Topic 104, we will cover the Filesystem Hierarchy Standard (FHS). For now, we note that local programs should have executables stored in the /usr/local tree in /usr/local/bin and man pages in /usr/local/man. configure scripts are likely to have a `--prefix` option to specify the install location. If the program is not FHS-compliant you may need to specify this option when you run the configure script. If you are building a product to replace an installed product, you may need to install it with /opt or /usr as the prefix.

In addition to possibly specifying a prefix, you may find other options that relate to the location of specific components, such as `--mandir` or `--infodir` to specify the location of man and info pages respectively.

Once you have reviewed the possible options and identified any that you may need to change, run the `configure` script, adding any options you need. Remember to add `./` before the configure command as your project directory will probably not be in your path. For example, you might run

```
./configure
or
./configure --prefix /usr/local
```

When you run configure, You will usually see messages indicating what type of system you have and what required tools are present or not present. If all goes well, you should have a Makefile built at the end of the configure process.

config.cache

When the configure script completes, it will store information about the configuration in a file called *config.cache* which will be in the same directory as the configure

script.

If you need to run `./configure` again be sure to remove your config.cache file first (using the `rm` command), as configure will use the settings from config.cache if it is present and it will not recheck your system.

Listing 17 shows some of the output from the configure step for the Dr Geo package that we unpacked earlier.

Listing 17. Configure Dr Geo.

```
[ian@localhost ~]$ cd drgeo-1.1.0
[ian@localhost drgeo-1.1.0]$ ./configure | less
checking for XML::Parser... ok
checking for iconv... /usr/bin/iconv
checking for msgfmt... /usr/bin/msgfmt
checking for msgmerge... /usr/bin/msgmerge
checking for xgettext... /usr/bin/xgettext
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking for g++... g++
checking for C++ compiler default output file name... a.out
checking whether the C++ compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
...
checking for guile... /usr/bin/guile
checking for guile-config... no
configure: error: guile-config required but not found
```

The configure script checks for a number of graphics conversion programs are part of the netpbm package. There is a warning because one possible conversion program is not found on the system. There is also a warning that use of the /usr/local prefix will require root access (at the installation step). Since this is the first time that configure has been run, there are some error messages related to Makefiles that do not yet exist, but would exist if we ran configure again. Finally, the configure script reports success.

Make and Makefiles

Once the configuration step is complete, you should have a file named *Makefile* in your project directory. This is called a *make file* because a program called *make* is used to process it and build your program. You may also have several more make files in subdirectories.

A make file contains *rules*, which are the instructions that tell the make program how to build things. The file also contains *targets*, which tell the make program what to build. The make program analyzes the make file and determines the order in which items must be built. For example, if an executable is built from three object files, then the object files must be built before they can be linked into an executable. A make file may perform installation tasks as well as building your program. Make file targets

will usually be available for several functions, such as:

make
with no options will just build the program. Technically this will build a *default target*, which usually means just build the program from sources.

make install
will install the program you built. You may need root authority for this if you are installing in /usr/local.

make clean
will erase files created by the make process.

make all
is sometimes used to perform the bulk of the make file's function with a single target. .

Consult your project documentation to see if there are additional targets or additional things you might need to do.

Now that your main Makefile has been created, use the `make` command, usually with no options, to build the executables, man pages and other parts of the program. Depending on the speed of your system and the complexity of the build process make may take only a minute or tow, or it may take much longer for a complex project.

Sometimes your build may not work. Common reasons include:

- Missing prerequisite packages
- Wrong level of prerequisite packages
- Wrong value for some parameter that you should have passed to configure or make.
- Missing compiler.
- Bugs in the configure script or generated Makefile.
- Source code bugs.

In our Dr Geo example, one of these problems was found at the configuration step, but this is not always the case. As you gain more experience with Linux, you will usually be able to identify and fix these problems. Sometimes you may need to check for a FAQ or mailing list that supports the package. Other times you may need to identify what you are missing and install it.

Installation

If all went well with your build, you are ready to install. The build step will build all the files you need, but they are not yet located in the correct places, ready for use. For example, binaries need to be copied to /usr/local/bin, and man pages to /usr/local/man, and so on.

Unless you specified a `--prefix` option, quite a few files and directories will probably be copied to your /usr/local tree. You will need root authority to write to the /usr/local tree in your filesystem. If you are not already logged in as root, use the `su` command to gain root authority. You will be prompted for the root password. Then use `make install` to install your newly built program. Depending on the size of the program, the install may take several seconds up to a few minutes to complete. We show part of the output for installing Dr Geo in Listing 18.

Listing 18. Installing Dr Geo.

```
[ian@attic4 drgeo-1.1.0]$ su
Password:
[root@attic4 drgeo-1.1.0]# make install
Making install in po
make[1]: Entering directory `/home/ian/drgeo-1.1.0/po'
if test -n ""; then \
   /usr/local/share; \
else \
  /bin/sh ../mkinstalldirs /usr/local/share; \
fi
installing az.gmo as /usr/local/share/locale/az/LC_MESSAGES/drgeo.mo
installing ca.gmo as /usr/local/share/locale/ca/LC_MESSAGES/drgeo.mo
installing cs.gmo as /usr/local/share/locale/cs/LC_MESSAGES/drgeo.mo
installing da.gmo as /usr/local/share/locale/da/LC_MESSAGES/drgeo.mo
installing de.gmo as /usr/local/share/locale/de/LC_MESSAGES/drgeo.mo
installing el.gmo as /usr/local/share/locale/el/LC_MESSAGES/drgeo.mo
installing en_CA.gmo as /usr/local/share/locale/en_CA/LC_MESSAGES/drgeo.mo
installing en_GB.gmo as /usr/local/share/locale/en_GB/LC_MESSAGES/drgeo.mo
...
/usr/bin/install -c drgeo /usr/local/bin/drgeo
/bin/sh ./mkinstalldirs /usr/local/share/applications
 /usr/bin/install -c -m 644 drgeo.desktop /usr/local/share/applications/drgeo.desktop
make[2]: Leaving directory `/home/ian/drgeo-1.1.0'
make[1]: Leaving directory `/home/ian/drgeo-1.1.0'
[root@attic4 drgeo-1.1.0]# exit
exit
[ian@attic4 drgeo-1.1.0]$
```

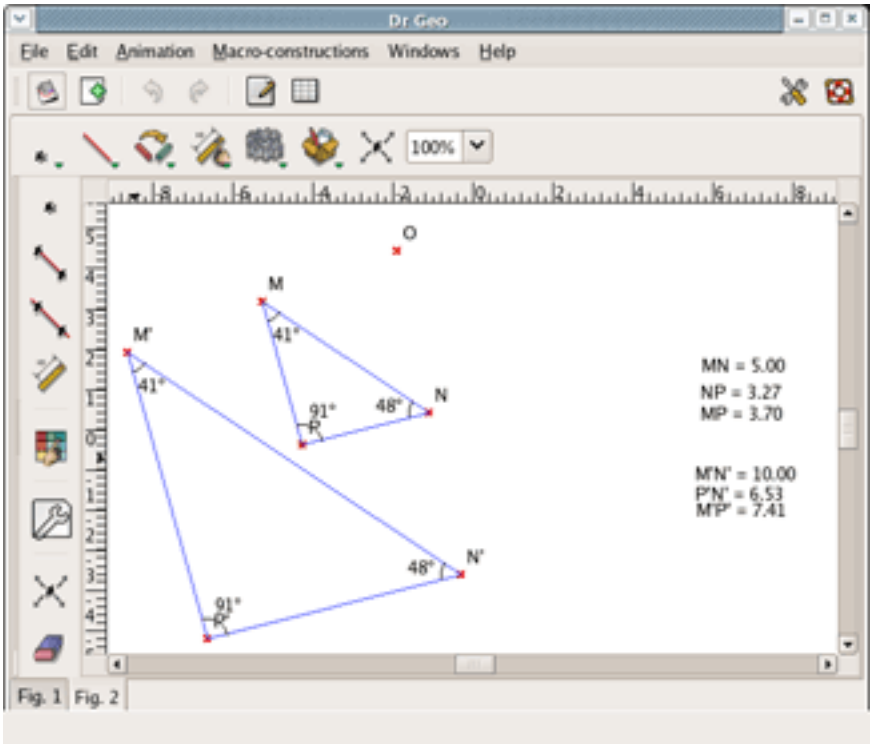
As well as copying files, make install should also make sure the installed files have the correct ownership and permissions. After the install finishes, the program is installed and ready for use, or possibly customize prior to using.

Note: It is very easy to make mistakes that can harm your system while you have root privileges, so remember to leave root mode by using the `exit` command or, in the bash shell, by pressing ctrl-d.

Run the program

If your program is now ready to run, you can try it out by typing the program name, `drgeo` in our example. Figure 1 shows a Dr Geo screen displaying one of the examples supplied with the program.

Figure 1. Running Dr Geo



Other things that you may need to do before running a program.

- Read the man page if one is part of the package. Try `man programname`.
- Customize configuration files, for example in /etc.
- Configure a program such as a server daemon to start automatically.

In this section we have covered relatively straight-forward installations from source. In the next few sections we'll talk more about libraries and library management as well as packages and how to install them.

Section 5. Manage shared libraries

This section covers material for topic 1.102.4 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 3.

In this section, you will learn how to determine the shared libraries that executable programs depend on. You will learn where system libraries are kept. We will cover installing packages, including shared libraries, in the next sections of this tutorial.

Static and dynamic executables

Linux systems have two types of executable program.

1. *Statically linked* executables contain all the library functions that they need to execute. All library functions are linked into the executable. They are complete programs that do not depend on external libraries to run. One advantage of statically linked programs is that they will work without installing prerequisites.

2. *Dynamically linked* executables are much smaller programs that are incomplete, in the sense that they require functions from external *shared* libraries in order to run. Besides being smaller, dynamic linking permits a package to specify prerequisite libraries without needing to include the libraries in the package. The use of dynamic linking also allows many running programs to share one copy of a library rather than occupying memory with many copies of the same code. For these reasons, most programs today use dynamic linking.

An interesting example on a typical Linux system is the `ln` command (/bin/ln) which creates links between files, either *hard* links, or *soft* (or *symbolic*) links. Shared libraries often involve symbolic links between a generic name for the library and a specific level of the library, so if the links aren't working, then the ln command might be inoperative. To protect against this possibility, Linux systems include a statically linked version of the ln program as the `sln` program (/sbin/sln). Listing 19 illustrates the great difference in size between these two programs.

Listing 19. Sizes of sln and ln.

```
[ian@lyrebird ian]$ ls -l /sbin/sln; ls -l /bin/ln
-rwxr-xr-x    1 root     root       457165 Feb 23  2005 /sbin/sln
-rwxr-xr-x    1 root     root        22204 Aug 12  2003 /bin/ln
```

The ldd command

Apart from knowing that a statically linked program is likely to be large, how do we tell whether a program is statically linked? And if it is dynamically linked, how do we know what libraries it needs? The answer to both questions is the `ldd` command which displays information about the library requirements of an executable. Listing 20 shows the output of the ldd command for the ln and sln executables.

Listing 20. Output of ldd for sln and ln.

```
[ian@lyrebird ian]$ ldd /sbin/sln /bin/ln
/sbin/sln:
        not a dynamic executable
/bin/ln:
        libc.so.6 => /lib/tls/libc.so.6 (0x00ebd000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x00194000)
```

Since ldd is actually concerned with dynamic linking, it tells us that sln is statically

linked by telling us that it is "not a dynamic executable", while it tells us the names of the two shared libraries (libc.so.6 and ld-linux.so.2) that the ln command needs, as well as where to find these libraries. Note that *.so* indicates that these are *shared objects* or dynamic libraries. In Listing 21 we use the `ls -l` command to show that these are indeed symbolic links to specific versions of the libraries.

Listing 21. Library symbolic links.

```
[ian@lyrebird ian]$ ls -l /lib/tls/libc.so.6; ls -l /lib/ld-linux.so.2
lrwxrwxrwx    1 root   root       13 May 18 16:24 /lib/tls/libc.so.6 -> libc-2.3.2.so
lrwxrwxrwx    1 root   root       11 May 18 16:24 /lib/ld-linux.so.2 -> ld-2.3.2.so
```

Dynamic loading

From the preceding you might be surprised to learn that ld-linux.so, which looks like a shared library, is actually an executable in its own right. This is the code that is responsible for dynamically loading. It reads the header information from the executable which is in the *Executable and Linking Format* or (*ELF*) format. From this information it determines what libraries are required and which ones need to be loaded. It then performs dynamic linking to fix up all the address pointers in your executable and the loaded libraries so that the program will run.

You won't find a man page for ld-linux.so, but you will find it mentioned under the man entry for ld.so, `man ld.so`. Listing 22 illustrates using the `--list` option of ld-linux.so to show the same information for the ln command that we showed with the ldd command in Listing 20.

Listing 22. Using ld-linux.so to display library requirements.

```
[ian@lyrebird ian]$ /lib/ld-linux.so.2 --list /bin/ln
        libc.so.6 => /lib/tls/libc.so.6 (0x00a83000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x00f2c000)
```

Note that the hex addresses are different between the two listings. They are also likely to be different if you run ldd twice.

Dynamic library configuration

So how does the dynamic loader know where to look for executables? As with many things on Linux, there is a configuration file in /etc. In fact, there are two configuration files, /etc/ld/so/conf and /etc/ld.so.cache. Listing 23 shows the contents of /etc/ld.so.conf on two different systems. Note that on the attic4 system (running fedora Core 4), /etc/ld.so.conf specifies that all the .conf files from the subdirectory ld.so.conf.d should be included. The actual contents of /etc/ld.so.conf may be different on your system.

Listing 23. Content of /etc/ld.so.conf.

```
[ian@lyrebird ian]$ cat /etc/ld.so.conf
/usr/kerberos/lib
/usr/X11R6/lib
/usr/lib/qt-3.1/lib
[
[ian@attic4 ~]$ cat /etc/ld.so.conf
include ld.so.conf.d/*.conf
```

Loading of programs needs to be fast, so the ld.so.conf file is processed to the `ldconfig` command to process all the libraries from ld.so.conf.d as well as those from the trusted directories, /lib and /usr/lib. The dynamic loader uses the ld.conf.cache file to locate files that are to be dynamically loaded and linked. If you change ld.co.conf (or add new included files to ld.so.conf.d you must run the `ldconfig` command (as root) to rebuild your ld.conf.cache file.

Normally, you use the ldconfig command without parameters to rebuild ld.so.cache. There are several other parameters you can specify to override this default behavior. As usual, try `man ldconfig` for more information. We illustrate the use of the `-p` parameter to display the contents of ld.so.cache in Listing 24.

Listing 24. Using ldconfig to display ld.so.cache.

```
[ian@lyrebird ian]$ /sbin/ldconfig -p | more
768 libs found in cache `/etc/ld.so.cache'
        libzvt.so.2 (libc6) => /usr/lib/libzvt.so.2
        libz.so.1 (libc6) => /usr/lib/libz.so.1
        libz.so (libc6) => /usr/lib/libz.so
        libx11globalcomm.so.1 (libc6) => /usr/lib/libx11globalcomm.so.1
        libxsltbreakpoint.so.1 (libc6) => /usr/lib/libxsltbreakpoint.so.1
        libxslt.so.1 (libc6) => /usr/lib/libxslt.so.1
        libxmms.so.1 (libc6) => /usr/lib/libxmms.so.1
        libxml2.so.2 (libc6) => /usr/lib/libxml2.so.2
        libxml2.so (libc6) => /usr/lib/libxml2.so
        libxmltok.so.0 (libc6) => /usr/lib/libxmltok.so.0
        libxmlparse.so.0 (libc6) => /usr/lib/libxmlparse.so.0
        libxml.so.1 (libc6) => /usr/lib/libxml.so.1
        libxerces-c.so.24 (libc6) => /usr/lib/libxerces-c.so.24
        ...
        lib-gnu-activation-20030319.so (libc6) => /usr/lib/lib-gnu-activation-20030319.so
        ld-linux.so.2 (ELF) => /lib/ld-linux.so.2
```

If you're running an older application that needs a specific older version of a shared library, or if you're developing a new shared library or version of a shared library, you might want to override the default search paths used by the loader. This may also be needed by scripts that use product-specific shared libraries that are installed in the /opt tree.

Just as you can set the PATH variable to specify a search path for executables, you can set the LD_LIBRARY_PATH variable to a colon separated list of directories that should be searched for shared libraries before the system ones specified in ld.so.cache. For example, you might use a command like

```
export
LD_LIBRARY_PATH=/usr/lib/oldstuff:/opt/IBM/AgentController/lib
```

In the remaining sections of this tutorial, we will look at package management.

Section 6. Debian package management

This section covers material for topic 1.102.5 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 8.

In an earlier section we learned about installing programs from source. In this section you will learn about another alternative used by most distributions today, *package management*, in which prebuilt programs or sets of programs are distributed as a *package*, ready for installation on a particular distribution. In this section and the next we will look at package management, focusing on two widely used package management systems. These are the *Advanced Packaging Tool* or *APT* developed by Debian and the *Red Hat Package Manager* or *RPM* developed by Red Hat.

Package management overview

In the Dr Geo example of the earlier section, our configuration step failed initially because we did not have a particular prerequisite program. Package management tools formalize the notion of prerequisites and versions and standardize file locations on your system, as well as providing a tracking mechanism that helps you determine what packages are installed. The result is easier software installation, maintenance and removal.

While you may still want to install programs from source for the reasons enumerated in the previous section, you will probably do most of your system maintenance and program installation using the package manager that was set up for your distribution.

From a user perspective, the basic package management function is provided by commands. As Linux developers have striven to make Linux easier to use, the basic tools have been supplemented by other tools, including GUI tools which hide some of the complexities of the basic tools from the end user. In these two sections we focus on the basic tools, although we will mention some of the other tools so that you have a starting place to learn about them.

Installing Debian packages

Let us return to the problem we encountered earlier with the Dr Geo program source. As it happens, that problem occurred on a Fedora Core 4 system which uses RPM package management. Fortunately for this section of the tutorial, I was also missing some guile components on a Debian-based Ubuntu system where I tried installing Dr Geo. The error this time is shown in Listing 25.

Listing 25. Missing guile function.

```
ian@attic4:~$ cd drgeo-1.1.0
ian@attic4:~/drgeo-1.1.0$ ./configure
checking for perl... /usr/bin/perl
checking for XML::Parser... ok
checking for iconv... /usr/bin/iconv
checking for msgfmt... /usr/bin/msgfmt
...
checking for guile... no
configure: error: guile required but not found
i
```

The package we need is the guile package. We can install that using the `apt-get` command as shown in Listing 26. Note the use of the `sudo` command which is the usual Ubuntu method of doing work with root authority.

Listing 26. Installing guile using apt-get.

```
ian@attic4:~$ sudo apt-get install guile
Reading package lists... Done
Building dependency tree... Done
Note, selecting guile-1.6 instead of guile
Suggested packages:
  guile-1.6-doc
The following NEW packages will be installed:
  guile-1.6
0 upgraded, 1 newly installed, 0 to remove and 24 not upgraded.
Need to get 31.5kB of archives.
After unpacking 209kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com hoary/main guile-1.6 1.6.7-1ubuntu1 [31.5kB]
Fetched 31.5kB in 0s (37.4kB/s)

Preconfiguring packages ...
Selecting previously deselected package guile-1.6.
(Reading database ... 84435 files and directories currently installed.)
Unpacking guile-1.6 (from .../guile-1.6_1.6.7-1ubuntu1_i386.deb) ...
Setting up guile-1.6 (1.6.7-1ubuntu1) ...
i
```

From the output we see that apt-get has read a package list from somewhere (more on that shortly), built a dependency tree, determined that guile-doc is recommended for installation with guile, and downloaded the guile package from the Internet. The guile package has then been unpacked, installed and set up. Note that the extension used for Debian packages is .deb. The full file name of our guile package is guile-1.6_1.6.7-1ubuntu1_i386.deb.

If apt-get notices that the package you are trying to install depends on other packages, it will automatically fetch and install those as well. In our example, only guile was installed, because all dependencies were already satisfied. Based on advice in the output, we could install guile-doc (or guile-1.6.doc).

Suppose that, instead of installing guile-doc, we wanted to find out whether the installation of guile-doc depends on other packages. We can use the `-s` (for *simulate*) option on apt-get. There are several other options with equivalent function, such as `--just-print` and `\--dry-run`. check the man pages for full details. Not surprisingly, the documentation for a package we have just installed doesn't' have any prerequisites, so in Listing 27 we illustrate a slightly more useful example with a

simulated install of the ssl-cert package which requires the openssl package.

Listing 27. Simulated or dry-run install of ssl-cert.

```
ian@attic4:~$ sudo apt-get -s install ssl-cert
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
  openssl
Suggested packages:
  ca-certificates
The following NEW packages will be installed:
  openssl ssl-cert
0 upgraded, 2 newly installed, 0 to remove and 24 not upgraded.
Inst openssl (0.9.7e-3 Ubuntu:5.04/hoary)
Inst ssl-cert (1.0-11 Ubuntu:5.04/hoary)
Conf openssl (0.9.7e-3 Ubuntu:5.04/hoary)
Conf ssl-cert (1.0-11 Ubuntu:5.04/hoary)
```

We see that two new packages are required and we see the order in which they will be installed and configured.

Package resource list: apt-setup

We mentioned that apt-get read a package list from somewhere. That somewhere is /etc/apt/sources.list. This is a list that you can edit yourself, but you will probably prefer to set up using the `apt-setup` command. The `apt-setup` command is an interactive tool that knows the location of the main APT repositories. You can access package sources on a CD-ROM, your local file system, or over a network using http or ftp.

If your distribution installed a /etc/apt/sources.list file for you, it may not have your CD-ROM as a source for packages. This can be inconvenient, particularly in the early stages of experimenting with a new system when you might want to add many packages,most of which have not yet been updated. In this case, you can use the command

```
apt-cdrom add
```

to add your CD-ROM to the list of package sources.

Apt-get and the other tools we will learn about, use a local database to determine what packages are installed. They can check installed levels against available levels. To do this, information on available levels is retrieved from the sources listed in /etc/apt/sources.list and stored on your local system. If you update your /etc/apt/sources.list file, then you should run

```
apt-get update
```

This will bring your stored data about available package levels up-to-date. In general, you should always doing this before installing any new package.

Removing or upgrading packages.

Before leaving apt-get, we will mention two other useful options.

If you installed a package and later want to uninstall it, you use the `remove` option with apt-get. Listing 28 shows how to remove the guile package that we installed earlier.

Listing 28. Removing the guile package.

```
ian@attic4:~$ sudo apt-get remove guile
Reading package lists... Done
Building dependency tree... Done
Note, selecting guile-1.6 instead of guile
The following packages will be REMOVED:
  guile-1.6
0 upgraded, 0 newly installed, 1 to remove and 24 not upgraded.
Need to get 0B of archives.
After unpacking 209kB disk space will be freed.
Do you want to continue [Y/n]? Y
(Reading database ... 84455 files and directories currently installed.)
Removing guile-1.6 ...
```

The other option we will mention is the `upgrade` option. This option upgrades all the installed packages on your system to the latest levels. Do not confuse this with the `update` option which merely refreshes the information about available packages.

For more information on other capabilities and options for apt-get, see the man page.

The apt.conf file

If you check the man page for apt-get, you will find that there are many options. If you use the apt-get command a lot and you find the default options are not to your liking, you can set new defaults in /etc/apt/apt.conf. A program, `apt-config` is available for scripts to interrogate the apt.conf file. See the man pages for apt.conf and apt-config for further information.

Debian package information

We will now look at some tools for getting information about packages. Some of these tools also do other things, but we will focus here on the informational aspects.

Package status with dpkg

Another tool that is part of the APT system is the `dpkg` tool. This is a medium level package management tool which can install and remove packages as well as displaying status information. Configuration of dpkg may be controlled by /etc/dpkg/dpkg.cfg. Individual users may also have a .dpkg.cfg file in their home directory to provide further configuration. If you do not have either of these files check /usr/share/doc/dpkg/dpkg.cfg for an example.

The dpkg tool uses many files in the /var/lib/dpkg tree in your filesystem. In particular, the file /var/lib/dpkg/status contains status information about packages on

your system. Listing 29 shows the use of `dpkg -s` to display the status of the guile package after we installed it. Remember that we actually installed guile-1.6. We see in Listing 29 that we need to give the full form of the name, not just the abbreviated form.

Listing 29. Guile package status.

```
ian@attic4:~$ dpkg -s guile
Package `guile' is not installed and no info is available.

Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.
ian@attic4:~$ dpkg -s guile-1.6
Package: guile-1.6
Status: install ok installed
Priority: optional
Section: interpreters
Installed-Size: 204
Maintainer: Rob Browning <rlb@defaultvalue.org>
Architecture: i386
Version: 1.6.7-1ubuntu1
Provides: guile
Depends: guile-1.6-libs, libc6 (>= 2.3.2.ds1-4), libguile-ltdl-1
Suggests: guile-1.6-doc
Conflicts: libguile-dev (<= 1:1.4-24)
Description: The GNU extension language and Scheme interpreter
 Guile is a Scheme implementation designed for real world programming,
 providing a rich Unix interface, a module system, an interpreter, and
 many extension languages.  Guile can be used as a standard #! style
 interpreter, via #!/usr/bin/guile, or as an extension language for
 other applications via libguile.
```

Packages and the files in them

You will often want to know what is in a package or what package a particular file came from. These are both tasks for dpkg. Listing 30 illustrates the use of `dpkg -L` to list the files (including directories) installed by the guile package.

Listing 30. What is in the guile package?

```
root@attic4:~# dpkg -L guile-1.6
/.
/usr
/usr/bin
/usr/bin/guile-1.6-snarf
/usr/bin/guile-1.6-tools
/usr/bin/guile-1.6
/usr/bin/guile-1.6-config
/usr/share
/usr/share/guile
/usr/share/guile/1.6
/usr/share/guile/1.6/scripts
/usr/share/guile/1.6/scripts/autofrisk
/usr/share/guile/1.6/scripts/display-commentary
/usr/share/guile/1.6/scripts/doc-snarf
/usr/share/guile/1.6/scripts/frisk
/usr/share/guile/1.6/scripts/generate-autoload
/usr/share/guile/1.6/scripts/lint
/usr/share/guile/1.6/scripts/PROGRAM
/usr/share/guile/1.6/scripts/punify
/usr/share/guile/1.6/scripts/read-scheme-source
/usr/share/guile/1.6/scripts/snarf-check-and-output-texi
/usr/share/guile/1.6/scripts/snarf-guile-m4-docs
/usr/share/guile/1.6/scripts/use2dot
```

```
/usr/share/doc
/usr/share/doc/guile-1.6
/usr/share/doc/guile-1.6/copyright
/usr/share/doc/guile-1.6/changelog.Debian.gz
/usr/lib
/usr/lib/menu
/usr/lib/menu/guile-1.6
```

To find which package contains a specific file, use the `-S` option of dpkg, as shown in Listing 31. The name of the package is listed on the left.

Listing 31. What package contains a file?

```
ian@attic4:~$ dpkg -S /usr/share/guile/1.6/scripts/lint
guile-1.6: /usr/share/guile/1.6/scripts/lint
```

You may notice that the list in Figure 30 did not include /usr/bin/guile, yet the command `which guile` says that this is the program that will be run if you type `guile`. When this occurs you may need to do some extra sleuth work to find where a package comes from. For example, the installation set up step may perform tasks such as creating symbolic links that are not listed as part of the package contents. A recent addition to Linux systems is the *alternatives* system which is managed using the `update-alternatives` command. In Listing 32, we show how to use the `ls` command to see what the guile command is symbolically linked to. The link to the /etc/alternatives directory is a tip off that we are using the alternatives system, so we use the `update-alternatives` command to find more information and finally we can use the `dpkg -S` command to confirm that the guile command comes from the guile-1.6 package. The setup for the alternatives system would have been done by a post install script that is part of the guile-1.6 package.

Listing 32. A more complex use of dpkg -S

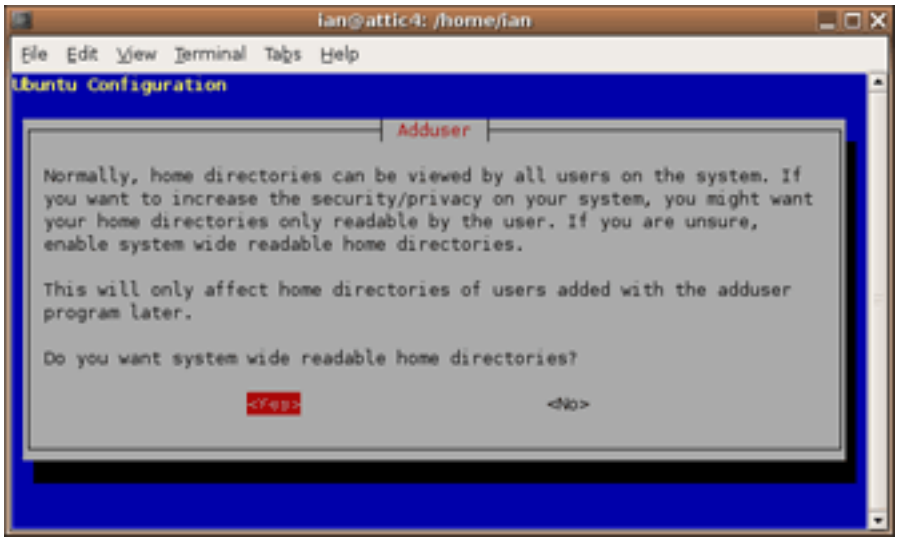
```
ian@attic4:~$ ls -l $(which guile)
lrwxrwxrwx  1 root root 23 2005-09-06 23:38 /usr/bin/guile -> /etc/alternatives/guile
ian@attic4:~$ update-alternatives --display guile
guile - status is auto.
 link currently points to /usr/bin/guile-1.6
/usr/bin/guile-1.6 - priority 160
 slave guile-config: /usr/bin/guile-1.6-config
 slave guile-snarf: /usr/bin/guile-1.6-snarf
 slave guile-tools: /usr/bin/guile-1.6-tools
Current `best' version is /usr/bin/guile-1.6.
ian@attic4:~$ dpkg -S /usr/bin/guile-1.6
guile-1.6: /usr/bin/guile-1.6
```

Reconfiguring Debian packages.

APT includes a capability called *debconf* which is used to configure packages after they are installed. Packages that use this capability (and not all do) can be reconfigured after they are installed. The easiest way to do this is to use the `dpkg-reconfigure` command. For example, the adduser command may create home directories that are readable by all system users. You may not want this for privacy reasons. Figure 2 illustrates the configuration question applicable to the

adduser package. Run `dpkg-reconfigure adduser` (as root) to generate this screen.

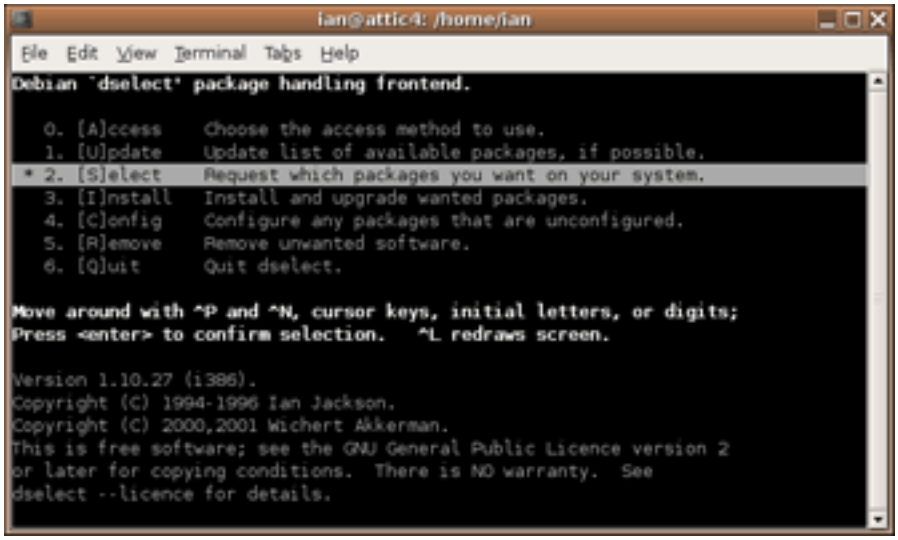
Figure 2. Using dpkg-reconfigure



Using dselect

Earlier, we mentioned that the status for packages is kept in /var/lib/dpkg/status. We also mentioned that dpkg could do more than just display package information. We will now take a brief look at the `dselect` command which provides a text-based full-screen interface (using ncurses) to the package management functions of dpkg. You can use dselect to install or remove packages as well as to control status flags that indicate whether packages should be kept up-to-date or held in their present state, for example. If you run the `dselect` command (as root) you will see a screen similar to that of Figure 3.

Figure 3. Running dselect



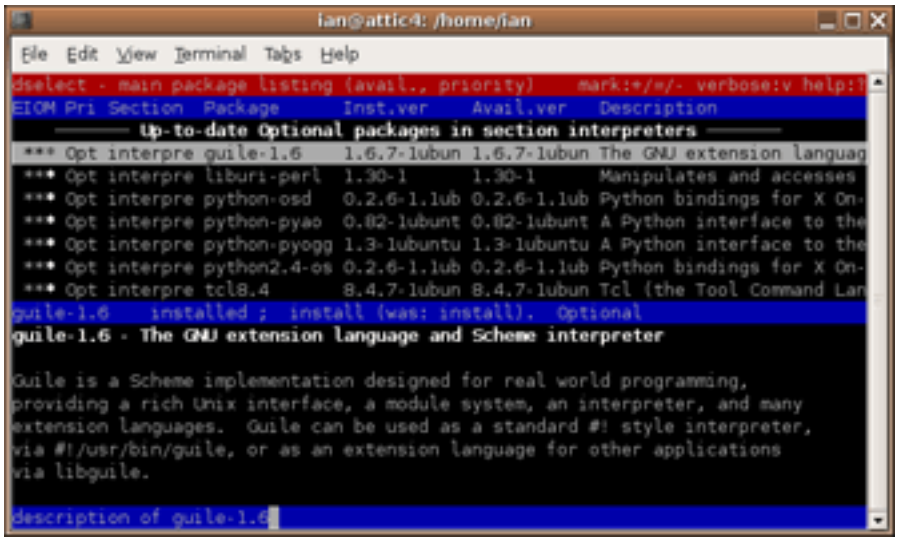
Using dselect Select mode

You can view and change the status of each package by choosing the Select option

that we highlighted in figure 3. You will then see a help screen. Press the space key to exit help at any time. You will then see a list of packages and package groups.

You can search for packages using / followed by a search string. Figure 4 shows the result of searching for "guile".

Figure 4. Selection screen for dselect



Package selection states

The selection state for each package can be seen under the cryptic heading *EIOM*. These letters stand for *Error*, *Installed state*, *Old mark*, and *Mark*. You may use the "v" key to toggle between the brief form of this display and a form that displays that states as words.

The fourth, or M, column is the one we look at now. This describes what will happen after we finish with the selection screen and move to the install screen. The marks have the following meanings:

*****
: Install or upgrade to the latest version

=
: Hold this package in its present state and version

- (hyphen)
: Delete the package but keep its configuration in case it is reinstalled later

_ (underscore)
: Delete this package and purge the configuration.

To change the mark, press the corresponding key, except that you press the "+" key to mark a package for install or upgrade. When you are finished, press **Enter** to confirm your package selection changes or press "X" (upper case X) to cancel without saving your changes.This will return you to the screen of Figure 3, with the Install option selected. Press **Enter** to install or upgrade your system.

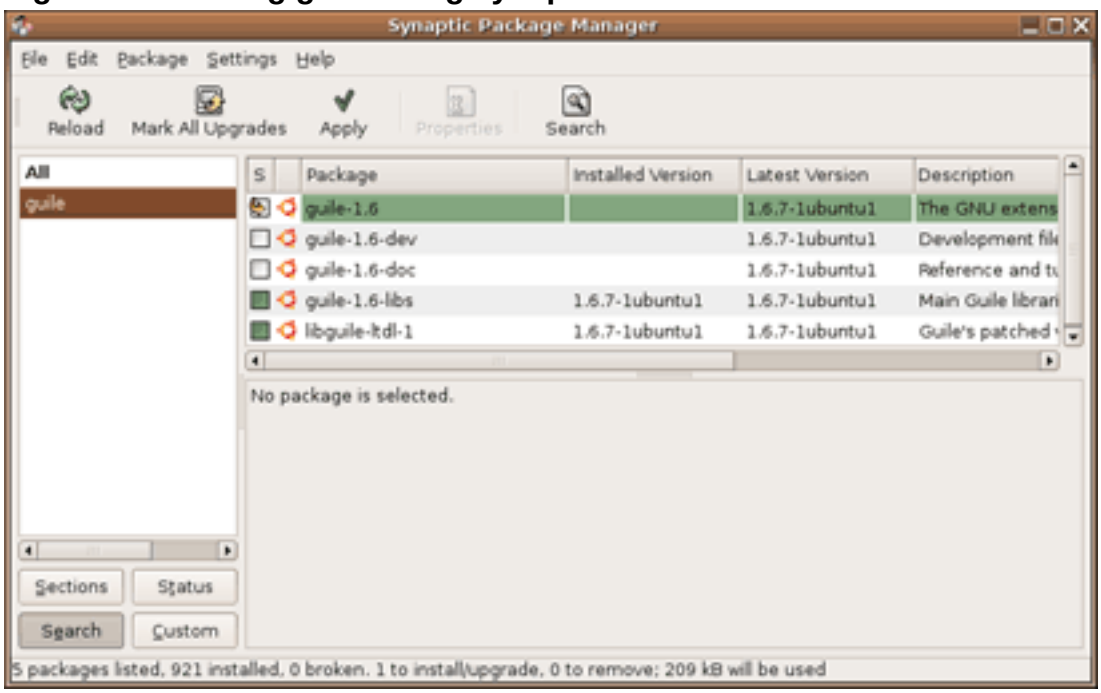
If you need help at any time, type "?" (question mark). Press the space to to get back out of help.

Upgrading Debian with other tools

We have now seen that dselect can help you install or remove individual packages as well as upgrade all the packages on your system to the latest level. If you want to do this from the command line, you can use `apt-get dselect-upgrade`, which honors the status marks that we saw set with dselect.

In addition to dselect, there are several other interactive package management interfaces for Debian systems, including aptitude, synaptic, gnome-apt and wajig. Synaptic is a graphical application for use with the X Window System. Figure 5 shows the synaptic user interface with our old friend, the guile package, marked for installation.

Figure 5. Installing guile using synaptic



The **Apply** button will install guile and update any other packages that are scheduled for update. The **Reload** button will refresh the package lists. If you are used to GUI interfaces, you may find synaptic easier to use than apt-get, dpkg or dselect.

Finding Debian packages

In our final topic on Debian package management, we will look at ways to find packages. Usually, apt-get and the other tools we have looked at will already know about any Debian package you might need from the list of available packages. A command that we haven't used yet is `apt-cache`, which is useful for searching

package information on your system. Apt-cache can search using regular expressions (which we will learn more about in the tutorial for Topic 103). Suppose we wanted to find the name of the package containing the Linux loader. Listing 33 shows how we might accomplish this.

Listing 33. Searching for the Linux loader with apt-cache

```
ian@attic4:~$ apt-cache search "linux loader"
lilo - LInux LOader - The Classic OS loader can load Linux and others
lilo-doc - Documentation for LILO (LInux LOader)
```

We saw earlier that dselect and synaptic also offer search tools. If you use synaptic, note that you have options with the search menu to specify whether to search only package names or package descriptions as well.

If you still can't find the package, you may be able to find it at package among the list of packages on the Debian site (see Resources), or elsewhere on the Internet.

If you do find and download a `.deb` file, you can install it using `dpkg -i`. For example, our Dr Geo example happens to be available as a .deb package from the official Debian package repository.

Listing 34. Installing Dr Geo from a .deb package

```
ian@attic4:~$ ls drg*.deb
drgeo_1-1.0.0-1_i386.deb
ian@attic4:~$ sudo dpkg -i drgeo_1-1.0.0-1_i386.deb
Password:
Selecting previously deselected package drgeo.
(Reading database ... 84435 files and directories currently installed.)
Unpacking drgeo (from drgeo_1-1.0.0-1_i386.deb) ...
Setting up drgeo (1.0.0-1) ...
```

Note that the source tarball for this package was at a higher level (1.1.0) than the deb package (1.0.0-1). If you installed Dr Geo and, for some reason, it did not work, you might need to try installing from source.

If all else fails, there is another possible source for packages. Suppose you find a program packaged as an RPM rather than a .deb. You may be able to use the `alien` program which can convert between package formats. You should read the alien documentation carefully as not all features of all package management systems can be converted to another format by alien.

There is a lot more to the Debian package management system than covered here. There is also a lot more to Debian than its package management system. See Resources for additional links.

Section 7. Red Hat Package Manager (RPM)

This section covers material for topic 1.102.6 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 8.

In the previous section on Debian package management, we gave you a brief package management overview. In this section we will focus on the *Red Hat Package Manager* or *RPM* developed by Red Hat. RPM and APT have many similarities. Both can install and remove packages. Both keep a database of installed packages. Both have basic command line functionality as well as other tools to provide more user-friendly interfaces. Both can retrieve packages from the Internet. In general, there are not as many programs to deal with RPM packages as there are for APT packages, although the `rpm` command has extensive capabilities. Another difference is that RPM does not maintain information about available packages on your system to the same extent that dpkg does.

Red Hat introduced RPM in 1995. RPM is now the package management system used for packaging in the Linux Standard Base (LSB). The rpm command options are grouped into three subgroups for:

- Querying and verifying packages
- Installing, upgrading and removing packages
- performing miscellaneous functions.

We will focus on the first two in this tutorial. You will find information about the miscellaneous functions in the man pages for rpm.

We should also note that *rpm* is the command name for the main command used with RPM, while *.rpm* is the extension used for RPM files. So "an rpm" or "the xxx rpm" will generally refer to an RPM file, while "rpm" will usually refer to the command.

Installing and removing RPM packages

As we did in the previous section, we'll look at the problems we encountered installing Dr Geo on a Fedora Core 4 system in the section Make and install programs. You may recall from Listing 17 that we were missing the guile-config command.

.

Getting started with rpm

The rpm command can install packages from local filesystems or from the Internet using either http or ftp. Listing 35 sows installation of the guile-devel package using the `rpm -ivh` command and a network source for the package.

Listing 35. Installing guile-devel with rpm

```
[root@attic4 ~]# rpm -ivh http://download.fedora.redhat.com/pub/fedora\
> /linux/core/4/i386/os/Fedora/RPMS/guile-devel-1.6.7-2.i386.rpm
```

```
Retrieving http://download.fedora.redhat.com/pub/fedora/linux/core/4/i386/os/Fedora/
RPMS/guile-devel-1.6.7-2.i386.rpm
Preparing...                ########################################### [100%]
   1:guile-devel            ########################################### [100%]
```

Note that the -v option gives verbose output and the -h option shows hash marks (#) to indicate progress. If you wanted to examine the package before installing from the network, you might want to download it first and then install. we'll talk about examining packages in a moment, but for now, let us use the `wget` command to retrieve the package and then install it from our local filesystem **without** using the `-vh` options. The output is shown in Listing 36.

Listing 36. Installing guile-devel from a file

```
[root@attic4 ~]# wget http://download.fedora.redhat.com/pub/fedora/\
> linux/core/4/i386/os/Fedora/RPMS/guile-devel-1.6.7-2.i386.rpm
--22:29:58--  http://download.fedora.redhat.com/pub/fedora/linux/core/4/i386/os/Fedora/
RPMS/guile-devel-1.6.7-2.i386.rpm
           => `guile-devel-1.6.7-2.i386.rpm'
Resolving download.fedora.redhat.com... 209.132.176.221
Connecting to download.fedora.redhat.com[209.132.176.221]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 481,631 [application/x-rpm]

100%[====================================>] 481,631      147.12K/s    ETA 00:00

22:30:02 (140.22 KB/s) - `guile-devel-1.6.7-2.i386.rpm' saved [481,631/481,631]

[root@attic4 ~]# ls guil*
guile-devel-1.6.7-2.i386.rpm
[root@attic4 ~]# rpm -i guile-devel-1.6.7-2.i386.rpm
```

No hash marks and no messages.

Re-installing an rpm

If you performed the above commands yourself, you would have seen an error on the second installation (or the first if you already had guile-devel installed) as it would have reported that guile-devel was already installed. To get around this, you should use the `-e` option to remove (or erase) the rpm before reinstalling it as shown in Listing 37. This would also apply if you needed to reinstall an rpm because you accidentally deleted some of its files.

Listing 37. Removing guile-devel

```
[root@attic4 ~]# rpm -e guile-devel
```

Forcibly installing an rpm

Sometimes removing an rpm isn't practical, particularly if there are other programs on the system that depend on it. For example, if you were trying to remove the guile package instead of the guile-devel package, you might see output like Listing 38, where many installed packages have dependencies on the guile package, so removal is not allowed.

Listing 38. Attempting to remove guile.

```
[root@attic4 ~]# rpm -q -R guile-devel
/bin/sh
/usr/bin/guile
guile = 5:1.6.7
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
[root@attic4 ~]# rpm -e guile
error: Failed dependencies:
        libguile-ltdl.so.1 is needed by (installed) g-wrap-1.3.4-8.i386
        libguile-ltdl.so.1 is needed by (installed) gnucash-1.8.11-3.i386
        libguile.so.12 is needed by (installed) g-wrap-1.3.4-8.i386
        libguile.so.12 is needed by (installed) gnucash-1.8.11-3.i386
        libqthreads.so.12 is needed by (installed) g-wrap-1.3.4-8.i386
        libqthreads.so.12 is needed by (installed) gnucash-1.8.11-3.i386
        guile is needed by (installed) g-wrap-1.3.4-8.i386
        guile = 5:1.6.7 is needed by (installed) guile-devel-1.6.7-2.i386
        /usr/bin/guile is needed by (installed) guile-devel-1.6.7-2.i386
```

Needless to say, it is impractical to remove all the packages with dependencies in such a case. The answer is to forcibly install the rpm using the `--force` option. In Listing 39 we illustrate forcibly reinstalling guile-devel from the file we downloaded for Listing 36.

Listing 39. Installing guile-devel with --force option.

```
[root@attic4 ~]# rpm -ivh --force guile-devel-1.6.7-2.i386.rpm
Preparing...                ########################################### [100%]
   1:guile-devel            ########################################### [100%]
```

Forcibly removing an rpm

There is an alternative to forcible install with the `--force` option that you may need on some occasions. You can remove an rpm using the `--nodeps` option which disables the internal dependency checking. You should generally do this **only** if you know what you are doing and **only** if you intend to fix the dependency problems by reinstalling the package. An example might be if you needed to revert to an earlier level of a package for some reason and wanted to be sure that all vestiges of the later version had been removed. The command you would use to remove the guile package without dependency checks is

```
rpm -e --nodeps guile
```

It is also possible to use the `--nodeps` option when installing an rpm. Again, this is not recommended, but may sometimes be necessary:

Upgrading RPM packages

Now that you know how to install and remove an RPM, let's look at upgrading an RPM to a newer level. This is similar to installing, except the we use the `-U` or the the `-F` option instead of the `-i` option. The difference between these two options is that the `-U` option will upgrade an existing package **or** install the package if it is not

already installed, while the the `-F` option will only upgrade or *freshen* a package that is already installed. Because of this, the `-U` option is frequently used, particularly when the command line contains a list of RPMs. This way, uninstalled packages are installed while installed packages are upgraded. Listing 40 shows the effect of attempting to upgrade guile-devel to the level that it is already at and then removing it and retrying the upgrade (which now functions as an install).

Listing 40. Upgrading guile-devel.

```
[root@attic4 ~]# rpm -Uvh  guile-devel-1.6.7-2.i386.rpm
Preparing...                ########################################### [100%]
        package guile-devel-1.6.7-2 is already installed
[root@attic4 ~]# rpm -e guile-devel
[root@attic4 ~]# rpm -Uvh  guile-devel-1.6.7-2.i386.rpm
Preparing...                ########################################### [100%]
   1:guile-devel            ########################################### [100%]
```

Querying RPM packages

You may have noticed in our examples that installing an rpm requires the full name of the package file (or URL), such as guile-devel-1.6.7-2.i386.rpm. On the other hand, removing the rpm requires only the package name, such as guile-devel. As with APT, RPM maintains an internal database of your installed packages, allowing you to manipulate installed packages using the package name. In this part of the tutorial, we will look at some of the information that is available to you from this database using the `-q` (for *query*) option of the rpm command.

The basic query simply asks if a package is installed. Add the `-i` option and you will get information about the package. Note that you need to have root authority to install, upgrade or remove packages, but non-root users can perform queries against the rpm database.

Listing 41. Displaying information about guile-devel.

```
[ian@attic4 ~]$ rpm -q guile-devel
guile-devel-1.6.7-2
[ian@attic4 ~]$ rpm -qi guile-devel
Name        : guile-devel                  Relocations: (not relocatable)
Version     : 1.6.7                            Vendor: Red Hat, Inc.
Release     : 2                             Build Date: Wed 02 Mar 2005 11:04:14 AM EST
Install Date: Thu 08 Sep 2005 08:35:45 AM EDT      Build Host: porky.build.redhat.com
Group       : Development/Libraries         Source RPM: guile-1.6.7-2.src.rpm
Size        : 1635366                          License: GPL
Signature   : DSA/SHA1, Fri 20 May 2005 01:25:07 PM EDT, Key ID b44269d04f2a6fd2
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Summary     : Libraries and header files for the GUILE extensibility library.
Description :
The guile-devel package includes the libraries, header files, etc.,
that you will need to develop applications that are linked with the
GUILE extensibility library.

You need to install the guile-devel package if you want to develop
applications that will be linked to GUILE. You also need to install
the guile package.
```

RPM packages and files in them

You will often want to know what is in a package or what package a particular file came from. To list the files in the guile-devel package, use the `-ql`.option as shown in Listing 42. There are many files in this package, so we've only shown part of the output.

Listing 42. Displaying information about guile-devel.

```
[ian@attic4 ~]$ rpm -ql guile-devel
/usr/bin/guile-config
/usr/bin/guile-snarf
/usr/include/guile
/usr/include/guile/gh.h
/usr/include/guile/srfi
/usr/include/guile/srfi/srfi-13.h
/usr/include/guile/srfi/srfi-14.h
/usr/include/guile/srfi/srfi-4.h
/usr/include/libguile
/usr/include/libguile.h
...
```

You can restrict the files listed to just configuration files by adding the `-c` option to your query. Similarly, the `-d` option limits the display to just documentation files.

Querying package files

The above package query commands query the RPM database for installed packages. If you've just downloaded a package and want the same kind of information you can get this using the `-p` option (for *package file*) on your query along with specifying the package **file** name (as used for installing the package). Listing 43 repeats the queries of Listing 41 against the package file instead of the RPM database.

Listing 42. Displaying guile-devel package file information.

```
[ian@attic4 ~]$ rpm -qp guile-devel-1.6.7-2.i386.rpm
guile-devel-1.6.7-2
[ian@attic4 ~]$ rpm -qpi guile-devel-1.6.7-2.i386.rpm
Name        : guile-devel                  Relocations: (not relocatable)
Version     : 1.6.7                             Vendor: Red Hat, Inc.
Release     : 2                          Build Date: Wed 02 Mar 2005 11:04:14 AM EST
Install Date: (not installed)            Build Host: porky.build.redhat.com
Group       : Development/Libraries      Source RPM: guile-1.6.7-2.src.rpm
Size        : 1635366                       License: GPL
Signature   : DSA/SHA1, Fri 20 May 2005 01:25:07 PM EDT, Key ID b44269d04f2a6fd2
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Summary     : Libraries and header files for the GUILE extensibility library.
Description :
The guile-devel package includes the libraries, header files, etc.,
that you will need to develop applications that are linked with the
GUILE extensibility library.

You need to install the guile-devel package if you want to develop
applications that will be linked to GUILE. You also need to install
the guile package.
```

Querying all installed packages

The `-a` option applies your query to all installed packages. This can generate a lot of output, so you will usually use it in conjunction with one or more filters, such as `sort`

to sort the listing, `more` or `less` to page through it, `wc` to obtain package or file counts, or `grep` to search for packages if you aren't sure of the name. Listing 43 shows the following queries:

1. A sorted list of all packages on the system
2. A count of all packages on the system.
3. A count of all files in all packages on the system.
4. A count of all documentation files installed with RPMs.
5. A search for all packages with "guile" (case-insensitive) as part of their name.

Listing 43. Queries against all packages.

```
[ian@attic4 ~]$ rpm -qa | sort | more
4Suite-1.0-8.b1
a2ps-4.13b-46
acl-2.2.23-8
acpid-1.0.4-1
alchemist-1.0.36-1
alsa-lib-1.0.9rf-2.FC4
alsa-utils-1.0.9rf-2.FC4
...
[ian@attic4 ~]$ rpm -qa | wc -l
874
[ian@attic4 ~]$ rpm -qal | wc -l
195681
[ian@attic4 ~]$ rpm -qald | wc -l
31881
[ian@attic4 ~]$ rpm -qa | grep -i "guile"
guile-devel-1.6.7-2
guile-1.6.7-2
```

Using `rpm -qa` can ease the administration of multiple systems. If you redirect the sorted output to a file on one machine, then do the same on the other machine, you can use the `diff` program to find differences.

Finding the owner for a file

Given that you can list all packages and all files in a package, you now have all the information you need to find which package owns a file. However, the rpm command provides a `-f` option to help you locate the package that owns a file. In our Dr Geo example in the Make and install programs section, we needed guile-config. Now that we have the guile-devel package installed, this is a an executable file on our path. Listing 44 shows how to use the `which` command to get the full path to the guile-config command, and a handy tip for using this output as input to the `rpm -qf` command. Note that the tick marks surrounding `which guile-config` are back-ticks. Another way of using this i bash is to use $(which guile-config)

Listing 44. Which package owns guile-config.

```
[ian@attic4 ~]$ which guile-config
/usr/bin/guile-config
[ian@attic4 ~]$ rpm -qf `which guile-config`
guile-devel-1.6.7-2
[ian@attic4 ~]$ rpm -qf $(which guile-config)
guile-devel-1.6.7-2
```

RPM dependencies

We saw earlier that we could not erase the guile package because of *dependencies*. In addition to files, an RPM package may contain arbitrary *capabilities* which other packages may depend on. In our example many other packages required capabilities provided by the guile package. And we could not have installed guile-devel if we had not already installed guile on our system. And once guile-devel is installed, it provides yet another reason why guile cannot be removed.

Usually, this all works out fine. If you need to install several packages at once, some of which may depend on others, simply give the whole list to your `rpm -Uvh` command and it will analyze the dependencies and perform the installs in the right order.

Besides trying to erase an installed package and getting an error message, the rpm command provides an option to interrogate installed packages or package files to find out what capabilities they depend on or *require*.This is the `--requires` option, which may be abbreviated to `-R`. Listing 45 shows the capabilities required by guile-config. Add the `-p` option and use the full RPM file name if you want to query the package file instead of the RPM database.

Listing 45. What does guile-config require.

```
[ian@attic4 ~]$ rpm -qR guile-devel
/bin/sh
/usr/bin/guile
guile = 5:1.6.7
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
```

In addition, to find out out what capabilities a package requires, you can find out what packages require a given capability (as is done when you attempt to remove a package). Listing 46 illustrates this for two of the capabilities required by guile-devel. Since this output may include duplicates, we've also shown you how to filter the output through `sort` and `uniq` to get each requiring package listed only once.

Listing 46. What needs /usr/bin/guile and guile.

```
[ian@attic4 ~]$ rpm -q --whatrequires /usr/bin/guile guile
guile-devel-1.6.7-2
g-wrap-1.3.4-8
guile-devel-1.6.7-2
[ian@attic4 ~]$ rpm -q --whatrequires /usr/bin/guile guile | sort|uniq
guile-devel-1.6.7-2
g-wrap-1.3.4-8
```

RPM package integrity.

To ensure the integrity of RPM packages, they will include a MD5 or SHA1 digest and they may be digitally signed. Packages that are digitally signed need a public key for verification. To check the integrity of an RPM package file use the `--checksig` (abbreviated to `-K`) option of rpm. You will usually find it useful to add the `-v` option for more verbose output.

Listing 47. Checking the integrity of the guile-devel package file.

```
[ian@attic4 ~]$ rpm --checksig guile-devel-1.6.7-2.i386.rpm
guile-devel-1.6.7-2.i386.rpm: (sha1) dsa sha1 md5 gpg OK
[ian@attic4 ~]$ rpm -Kv guile-devel-1.6.7-2.i386.rpm
guile-devel-1.6.7-2.i386.rpm:
    Header V3 DSA signature: OK, key ID 4f2a6fd2
    Header SHA1 digest: OK (b2c61217cef4a72a8d2eddb8db3e140e4e7607a1)
    MD5 digest: OK (cf47354f2513ba0c2d513329c52bf72a)
    V3 DSA signature: OK, key ID 4f2a6fd2
```

You may get an output line like:

```
      V3 DSA signature: NOKEY, key ID 16a61572
```

This means that you have a signed package, but you do not have the needed public key in your RPM database. Note that earlier versions of RPM may present the verification differently.

If a package is signed and you want to verify it against a signature, then you will need to locate the appropriate signature file and import it into your RPM database. You should first download the key and then check its fingerprint before importing it using the `rpm --import` command. For more information see the rpm man pages. You will also find more information on signed binaries at www.rpm.org.

Verifying an installed package

Similarly to checking the integrity of an rpm, you can also check the integrity of your installed files using `rpm -V`. This step makes sure that the files haven't been modified since they were installed from the rpm. As shown in Listing 49 there is no output from this command if the package is still good.

Listing 48. Verifying the installed guile-devel package.

```
[ian@attic4 ~]$ rpm -V guile-devel
```

Let us become root and remove /usr/bin/guile-config altogether and replace /usr/bin/guile-snarf with a copy of /bin/bash and try again. The results are shown in Listing 49.

Listing 48. Tampering with the guile-devel package.

```
[root@attic4 ~]# rm /usr/bin/guile-config
rm: remove regular file `/usr/bin/guile-config'? y
[root@attic4 ~]# cp /bin/bash /usr/bin/guile-snarf
cp: overwrite `/usr/bin/guile-snarf'? y
[root@attic4 ~]# rpm -V guile-devel
missing     /usr/bin/guile-config
S.5....T    /usr/bin/guile-snarf
```

This output shows us that the /usr/bin/guile-snarf fails MD5 sum, file size, and mtime tests. You can repair this by erasing the package and reinstalling, or forcibly installing as we saw earlier. Expect an error message if you remove the package, since one of its files is now missing.

Configuring RPM

RPM rarely needs configuring. In older versions of rpm, you could change things in /etc/rpmrc to control run-time operation. In recent versions, the file has been moved to /usr/lib/rpm/rpmrc where it is automatically replaced when the rpm package is upgraded, thus losing any changes you may have made. If any per-system configuration is required it may still be added to /etc/rpmrc, while per-user configuration should .rpmrc.in the user's home directory. You can find documentation on the format of these files in the book *Maximum RPM* (see Resources).

If you'd like to view the rpmrc configuration, there is even an rpm command option for that. Use the command:

```
rpm rpm --showrc
```

Repositories and other tools

By now you might be wondering where all these rpm packages come from, how you find them and how you manage updates to your system. If you have an RPM-based distribution, your distributor will likely maintain a *repository* of packages. Your distributor may also provide a tool for installing packages from the repository or updating your entire system. These tools may be graphical or command line or both. Some examples include:

- YaST (SUSE)
- up2date (Red Hat)
- yum - Yellow Dog Updater Modified (Fedora and others)
- Mandrake Software Management (Mandriva)

Usually these tools will handle multiple package updates in an automatic or semi-automatic fashion. They may also provide capabilities to display contents of repositories or search for packages. Consult the documentation for your distribution for more details.

If you can't find a particular RPM through your system tools, another good resource for locating RPMs is the Rpmfind.Net server (see Resources).

The next tutorial in this series covers Topic 103 -- GNU and Unix Commands.

Resources

Learn

- At the LPIC Program, find task lists, sample questions, and detailed objectives for each level of the Linux Professional Institute's Linux system administration certification.
- The Linux documentation project is the home of lots of useful Linux documentation, including the Linux Partition HOWTO on planning and creating partitions on IDE and SCSI hard drives.
- Learn more about FHS at the Filesystem Hierarchy Standard home page.
- Learn about GRUB legacy and GRUB 2 at the GNU GRUB home page. GNU GRUB is a multiboot boot loader derived from GRUB, GRand Unified Bootloader.
- The Multiboot Specification for an interface allows any complying boot loader to load any complying operating system.
- The Debian home page is your starting point for information about the Debian distribution.
- Refer to the "Debian installation guide" for more information on installing Debian.
- Display and search Debian package lists at Debian packages.
- Learn about the Debian package management utility, APT, in the "APT HOWTO."
- Get the Alien package converter.
- Check out the "Guide to creating your own Debian packages."
- Visit Ubuntu, Linux for Human Beings.
- At www.rpm.org find current information on the RPM software packaging tool and pointers to more information on RPM.
- The book *Maximum RPM* offers a comprehensive and systematic treatment of all aspects of RPM. It's available in both hard and soft copy formats.
- Read the RPM HOWTO to learn about the Red Hat Package Manager, RPM.
- Search for RPMs for your distribution at Rpmfind.Net and RPM Search.
- At the LSB Home, learn about the Linux Standard Base (LSB), a Free Standards Group (FSG) project to develop a standard binary operating environment.
- Find certification materials in book form in *LPI Linux Certification in a Nutshell* (O'Reilly, 2001).
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Download the System rescue CD-Rom, one of many tools available online to help you recover a system after a crash.
- Browse SourceForge.net, a large repository of open source projects for many platforms.
- Dr. Geo, interactive geometry is a source project used as an example in this tutorial.
- Go to Info-ZIP to get Zip and Unzip programs for Linux and other platforms.
- The Sphere Eversion Program shows you how to turn a sphere inside out without tearing or creasing it. The counterintuitive result behind this program is described in a video called "*Outside In*".
- Get Debian packages from the Debian home page.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Ian Shields
Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. You can contact Ian at ishields@us.ibm.com.